Communicating Process Architectures 2013
P.H. Welch et al. (Eds.)
Open Channel Publishing Ltd., 2013

Successful Termination in Timed CSP

Paul HOWELLS [a,1] and Mark D'INVERNO [b]
[a] *Department of Computer Science & Software Engineering, University of Westminster, UK*
[b] *Department of Computing, Goldsmiths, University of London, UK*

Abstract. In previous work the authors investigated the inconsistencies of how successful termination was modelled in Hoare, Brookes and Roscoe's original CSP. This led to the definition of a variant of CSP, called CSP$_T$. CSP$_T$ presents a solution to these problems by means of adding a termination axiom to the original process axioms. In this paper we investigate how successful process termination is modelled in Reed and Roscoe's Timed CSP, which is the temporal version of Hoare's original untimed CSP. We discuss the issues that need to be considered when selecting termination axioms for Timed CSP, based on our experiences in defining CSP$_T$. The outcome of this investigation and discussion is a collection of candidate successful termination axioms that could be added to the existing Timed CSP models, leading to an improved treatment of successful termination within the Timed CSP framework. We outline how these termination axioms would be added to the family of semantic models for Timed CSP. Finally, we outline what further work needs to be done once these new models for Timed CSP have been defined. For example, it would then be possible to define timed versions of the new more flexible parallel operators introduced in CSP$_T$.

Keywords. concurrency, CSP, Timed CSP, CSP$_T$, process termination

Introduction

In the original *failure-divergence* semantic model for *Communicating Sequential Processes* (CSP) [1,2,3] the incomplete treatment of successful process termination, and in particular parallel termination, permitted intuitively contradictory processes to be defined. For instance, it is possible to define a parallel process that appears to terminate several times before actually doing so – see Section 1 for an example.

Several alternative solutions have been proposed for these problems associated with modelling termination. The solution due to Roscoe [4,5] is seen as the "standard" solution. However, his solution requires termination to be viewed as a new kind of event, known as a "signal", distinct from other events; and only allows asynchronous parallel termination semantics.

Our response was to develop an alternative solution, in the form of a variant of CSP called CSP$_T$ [6,7]. CSP$_T$ solves the original termination problems and introduces three distinct, but related, parallel operators that between them provide a transparent and intuitive means for specifying a range of termination behaviours for networks of processes. Our aim for these operators was to remain as consistent as possible with the original model defined for CSP, whilst giving the system designer *precision* over the extent of parallel interaction (by means of a parametrised synchronisation set) and *flexibility* over the type of parallel termination.

[1]Corresponding Author: *Paul Howells, Department of Computer Science & Software Engineering, University of Westminster, 115 New Cavendish St., London, W1W 6RU, UK.* `E-mail: P.Howells@westminster.ac.uk`.

In this paper, we continue our investigation of the successful termination of processes within the CSP framework, by considering how successful termination is or should be modelled within the timed version of CSP, known as Timed CSP. We begin this process by considering the issues that influence the choice of a process axiom to capture the notion of successful termination within both the untimed and timed CSP worlds.

Following on from this discussion, we present our proposals for how an improved treatment of successful termination can be incorporated into Timed CSP. To achieve this, we shall adopt a similar approach to that taken in formulating our solution to the problems associated with modelling termination in CSP, which led to the definition of CSP_T and introduction of a termination axiom. Since there are several semantic models for Timed CSP (Reed and Roscoe [8,9,10,11]), this requires the definition of a termination axiom for each model. Once this task is complete, our intention is to replace the collection of parallel operators available in these timed versions of CSP with timed versions of those introduced in CSP_T.

Outline of the Rest of the Paper

In Section 1, we provide a brief introduction to CSP_T. We provide an overview of Timed CSP in Section 2. In Section 3 we review the issues that influenced the choice of a termination axiom in CSP_T and present a collection of candidate termination axioms that we propose to add to the existing models of Timed CSP. Related work on improving the modelling of termination in Timed CSP is discussed in Section 4. Finally, we present our conclusions and suggestions for future work in Section 5. (The Appendix contains a summary of the notation and definitions used throughout the paper.)

1. An Introduction to CSP_T[1]

In this section[2],we provide an overview of CSP_T, for a complete description please see [6, 7,13]. (For a summary of the CSP and CSP_T notation and definitions used throughout the paper, see the Appendix.)

In Hoare, Brookes and Roscoe's original CSP [1,2,3,14] there was a well known problem concerning the incomplete treatment of parallel termination by two of original parallel operators: alphabetised $(_A\|_B)$[3] and asynchronous/interleaving ($|||$). In particular, the type of termination that occurred using these operators was inconsistent and could vary depending on the termination of the processes being combined.

We shall now give an example to illustrate the type of problems that could arise. First recall, that the *successful termination* of a process in CSP (and CSP_T) is modelled by the event *tick* ($\checkmark$). Now consider the following process equivalence derivable within the original CSP:

$$(a \to SKIP) ||| (b \to SKIP) \equiv (a \to ((\checkmark \to b \to SKIP) \,\Box\, (b \to \checkmark \to SKIP))) \,\Box\, (b \to ((a \to \checkmark \to SKIP) \,\Box\, (\checkmark \to a \to SKIP)))$$

Clearly the $\checkmark$s on the right hand side cannot be interpreted as the successful termination of the process $(a \to SKIP) ||| (b \to SKIP)$, since it continues to perform a, b and $\checkmark$ events. This

[1]This section contains material extracted from Sections 3.1, 5.2 and 5.6 of [6] and Sections 3.2, 3.3 and 3.4 of [7] by permission of Springer and Elsevier, respectively.

[2]This section is identical to Section 1 of [12], a companion paper in these same Proceedings (CPA 2013). This duplication, by permission of the Editors, is to let both papers be self-contained.

[3]Here and henceforth, A and B represent the alphabets of the processes being composed. For example, as in $P_A\|_B Q$, where $A = \alpha(P)$ and $B = \alpha(Q)$ respectively.

illustrates how the original semantics for CSP could gave rise to these obviously undesirable and intuitively contradictory processes.

Several solutions have been proposed to this problem, including Tej and Wolf [15], Roscoe [4,5], Hoare and He [16] and the authors's own solution CSP_T [6]. For a detailed comparison of these solutions the interested reader is referred to [6].

Our starting point in defining CSP_T was the original failure-divergence model developed by Hoare, Brookes and Roscoe [2]. Our aim in modifying this model was to provide a more robust treatment of termination through the consistent and special handling of ✓ by the language (processes and operators) and semantics (failures and divergences).

For CSP_T, this was achieved by defining a new process axiom that captured our view of termination:

$$t \neq \langle\,\rangle \;\wedge\; (s^\frown\langle\checkmark\rangle^\frown t, \varnothing) \in F \;\Rightarrow\; s \in D \tag{T1}$$

where s and t are traces, F and D are the failure and divergence sets respectively of a process. This axiom means that if a process indicates that it has terminated (by means of the ✓) but continues to perform events (t), then it must have started diverging before it performed the ✓ (i.e. $s \in D$). For the rational behind this axiom, see Section 3.1.

This new termination axiom resolves the termination issues of the original semantics, and it is added to the existing CSP process axioms (D1) to (N5), see Appendix, to define CSP_T. Note that our view of tick (✓) is consistent with Hoare's, i.e. that it is a normal event, and have not adopted Roscoe's view that it is a special *signal* event. In doing this, we have defined a sub-model of the original failure-divergence model N_T such that, within this sub-model, all processes are well-behaved with respect to termination. In addition, three new forms of parallel operators were defined for CSP_T, each with a different form of termination semantics, as replacements for the original ones. We now introduce the language and model for CSP_T; we begin by introducing the three new parallel operators.

1.1. Parallel Operators of CSP_T

Our three new parallel operators are defined to be used as replacements for the original synchronous ($\|$), interleaving ($|||$) and alphabetised (${}_A\|_B$) parallel operators. It is necessary to define replacements for $|||$ and ${}_A\|_B$, as they do not satisfy (T1). These new parallel operators are *generalised* (or *interface*) style parallel operators, i.e. are parameterised by the set of events the processes are required to synchronise on. Each of these three operators has a distinct type of parallel termination semantics, and thus are distinct operators, see [7] for details. We call them *synchronous*, *asynchronous* and *race*, that we define here.

Synchronous: requires the successful termination of both P and Q; and the synchronisation of their termination, that is, ✓.

Asynchronous: requires the successful termination of both P and Q; and P and Q terminated asynchronously, i.e. they do not synchronise on ✓. (Roscoe [4,5] refers to this type of parallel termination semantics as *distributed termination*.)

Race: requires the successful termination of either P or Q asynchronously. Successful termination fails to occur only if both P and Q fail to terminate. Unlike synchronous and asynchronous termination where the environment observes a single ✓, under *race* termination semantics it can observe any ✓ that is performed by P or by Q. Consequently, the first ✓ the environment observes is taken as representing the termination of the parallel composition and whichever of P or Q did not terminate, i.e. did not preform the ✓, is aborted. Hence, with this type of termination semantics termination occurs as soon as either P or Q does so.

Note that in [4,5] Roscoe chooses to reject race termination semantics, due to the problems of dealing with the non-terminated process, e.g. the need for some powerful mechanism to manage its termination. We however, believe that it is preferable to at least offer system

specifiers the choice of using this type of termination semantics and let them resolve these issues, rather than ban it outright. We believe that this is not against the spirit of CSP, since there are other features available in CSP that also raise similar implementation issues, e.g. the various interrupt style operators.

The CSP_T parallel operators with these different forms of parallel termination semantics, are given in Table 1. Each operator is parametrised by a *synchronisation set* (Ω, Δ, Θ), that is the set of events on which the combined processes are required to synchronise. Events which are not in the synchronisation set but that can be performed by either P or Q or both are asynchronous events.

Table 1. CSP_T parallel operators.

Termination Semantics	Operator	Synchronisation Set	Notes
Generalised	$P\|\|_\Omega Q$	$\varnothing \subseteq \Omega \subseteq \Sigma$	
Synchronous	$P\|\|_\Delta Q$	$\{\checkmark\} \subseteq \Delta \subseteq \Sigma$	$\checkmark \in \Delta$
Asynchronous	$P\|\|\|_\Theta Q$	$\varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$	$\checkmark \notin \Theta$
Race	$P\|_\Theta Q$	$\varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$	$\checkmark \notin \Theta$

We define a generalised parallel operator (denoted $P\|_\Omega Q$) that is used to define the operators with synchronous and race termination semantics. To distinguish this from the synchronous termination operator we denote the synchronisation set by Ω rather than Δ. The synchronous parallel operator $\|_\Delta$ is simply the generalised one $\|_\Omega$, with the constraint that $\checkmark \in \Delta$, thus ensuring synchronous termination. The race termination operator $|_\Theta$ can also be defined using $\|_\Omega$ and *SKIP* as follows:

$$P|_\Theta Q \;\widehat{=}\; (P\|_\Theta Q); SKIP \qquad\qquad [\varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}]$$

where Σ is the set of all events and, for CSP_T, includes $\checkmark$. Note that CSP_T does not have the law $(P; SKIP = P)$, otherwise the above would mean the race and generalised operators were the same. To illustrate how this definition behaves, consider the process $(P\|_\Theta Q); R$, with $\Theta \subseteq \Sigma - \{\checkmark\}$. Whichever of the two $\checkmark$s R (the environment) observes first is taken as representing the termination of $P\|_\Theta Q$ and, hence, R proceeds to execute; whichever of P or Q did not terminate is aborted.

The asynchronous parallel operator $\|\|_\Theta$ is defined independently, as its form of termination semantics is not compatible with $\|_\Omega$, i.e. it cannot be used to define $\|\|_\Theta$. Both process operands must terminate for $\|\|_\Theta$ to terminate, so it is not the same as $\|_\Theta$ (see previous paragraph). Both processes must terminate for $\|_\Delta$, if $\Delta = \Theta \cup \{\checkmark\}$, but they *synchronise* on their $\checkmark$s (i.e. a non-terminating process can prevent the other operand from terminating, which is *not* the case for $\|\|_\Theta$).

Here are examples of the three parallel operators of CSP_T, that illustrate the difference between $|_\Theta$ and the two others (which for these examples, $\|\|_\varnothing$ and $\|_\varnothing$, are the same):

$$\begin{aligned} a \to SKIP \|_\varnothing b \to SKIP &\equiv a \to SKIP \|\|_\varnothing b \to SKIP \\ &\equiv (a \to b \to SKIP) \,\Box\, (b \to a \to SKIP) \\ a \to SKIP |_\varnothing b \to SKIP &\equiv (a \to (SKIP \sqcap (SKIP \,\Box\, b \to SKIP))) \\ &\quad \Box\, (b \to (SKIP \sqcap (SKIP \,\Box\, a \to SKIP))) \end{aligned}$$

It is important to note that we do not include the generalised parallel operator $\|_\Omega$ in the language of CSP_T, since this would defeat the purpose of the whole exercise. Since, it would again result in inconsistent processes, similar to the one given above in Section 1, with multiple ticks in its traces. However, by restricting its use to defining $\|_\Delta$ and $|_\Theta$ we ensure these forms of processes do not occur. Full details, including the operational semantics and semantic functions, can be found in [6,7].

1.2. The Model and Language of CSP_T

The model for CSP_T, N_T is defined by adding the Termination axiom (T1) to the process axioms (D1) to (N5) of the original CSP model N, given in the Appendix. Hence, in N_T the failure and divergence sets of a process are defined as for N, except that they also satisfy the process axiom (T1) as well as (D1) to (N5). The semantic functions $\mathcal{F}$ and $\mathcal{D}$ for N_T are the same as for N, and are given in [6].

The language for CSP_T is the same as that of the original CSP, but uses the more recent form of relational renaming instead of functional renaming and uses the three new parallel operators as replacements for $\|$, $|||$ and ${}_A\|_B$, see Table 2. It is defined as follows:

$$P ::= \bot \mid STOP \mid SKIP \mid a \rightarrow P \mid P \sqcap P \mid P \,\Box\, P \mid P; P \mid P\backslash a$$
$$\mid P[\![R]\!] \mid \mu p.F(p) \mid p \mid P \|_{\Delta} P \mid P |||_{\Theta} P \mid P |_{\Theta} P$$

where $a \in \Sigma - \{\checkmark\}$. $\bot$ is the divergent process (can be defined as $\mu p.p$). *STOP* is the deadlocked process and *SKIP* is the successfully terminating process. $a \rightarrow P$ is action prefix. $P \sqcap Q$ is nondeterministic choice and $P \,\Box\, Q$ is deterministic choice. $P; Q$ is sequential composition. $P\backslash a$ is event hiding. $P[\![R]\!]$ is action (relational) renaming, with the usual constraint [4,5] applying to the renaming relation R with respect to $\checkmark$, i.e. that no other event is mapped to it or that it is mapped to another event. p is a process variable, $\mu p.F(p)$ is recursion and in the definition of $F(p)$, only the above processes and operators can be used. $P \|_{\Delta} Q$, $P |||_{\Theta} Q$ and $P |_{\Theta} Q$ are the generalised synchronous, asynchronous and race parallel operators respectively. The processes and operators of CSP_T are *well-defined* and *well-behaved* in N_T.

Table 2. CSP_T replacements for $\|$, $|||$ and ${}_A\|_B$.

Termination Semantics	$\|\|\|$	${}_A\|_B$	$\|$
Synchronous ($\|_{\Delta}$)	$\Delta = \{\checkmark\}$	$\Delta = (A \cap B) \cup \{\checkmark\}$	$\Delta = (A \cup B) \cup \{\checkmark\}$
Asynchronous ($|||_{\Theta}$)	$\Theta = \varnothing$	$\Theta = (A \cap B) - \{\checkmark\}$	$\Theta = (A \cup B) - \{\checkmark\}$
Race ($|_{\Theta}$)	$\Theta = \varnothing$	$\Theta = (A \cap B) - \{\checkmark\}$	$\Theta = (A \cup B) - \{\checkmark\}$

1.3. Summary of Differences between CSP and CSP_T

The essential differences between Hoare, Brookes and Roscoe's original CSP and CSP_T can be summarised as follows:

- Termination axiom (T1) has been added to the original axioms (D1) - (N5). See Section 3.1 and the Appendix for details and explanation.
- Use relational renaming ($P[\![R]\!]$), instead of functional and inverse renaming.
- Replaced the interleaving ($|||$), synchronous ($\|$) and alphabetised (${}_A\|_B$) parallel operators with generalised synchronous ($\|_{\Delta}$), asynchronous ($|||_{\Theta}$) and race ($|_{\Theta}$) operators.

For a more detailed account of the development of CSP_T, the interested reader is referred to our previous work [6,7]. In the next section we provide an overview of the new concepts introduced for Timed CSP.

2. An Overview of Timed CSP

In this section we provide a brief introduction to the new notions introduced by moving from untimed to Timed CSP. We begin by outlining the different models for Timed CSP (Section 2.1), the timed versions of traces, refusals and failures and the new notion of stability (Section 2.2). Finally, we discuss the different forms of parallel operator used in the existing versions of Timed CSP and their termination behaviour (Section 2.3).

2.1. Timed CSP Models

Timed CSP was first introduced by Reed and Roscoe [8] and is the timed extension of Hoare's CSP [3]. The only extra component added to the original language of CSP was the delay process $WAIT\,t$, this process introduces an explicit delay of $t \geq 0$ time units (usually seconds). $WAIT\,t$ is a delayed form of *SKIP*, which successfully terminates after time t, where $WAIT\,0 = SKIP$.

Since the introduction of Timed CSP, it has been developed by the addition of different semantic models [9], additional operators [17], proof systems [18,19,20,21,22] and the notion of timewise refinement [23] and operational semantics [24]. For general introductions to Timed CSP see the work of Davies and Schneider [17,25,26,27].

There are several semantic models for Timed CSP, developed by Roscoe and Reed [8,9, 10,28,29,11]. The models form a hierarchy, see Figure 1, ordered by the information content of the observations that can be made in each model. The untimed models occupy the lowest position (the inner diamond of Figure 1) in the hierarchy; above them (the outer diamond) are the timed models. The lowest model is the Untimed Trace model M_T, and the highest (most complex and comprehensive) is the Timed Failure-Stability model TM_{FS}. The mathematics for the semantic models for untimed CSP have largely been based on lattice theory, whereas for Timed CSP it is topology [9,30], and in particular complete (ultra) metric spaces.

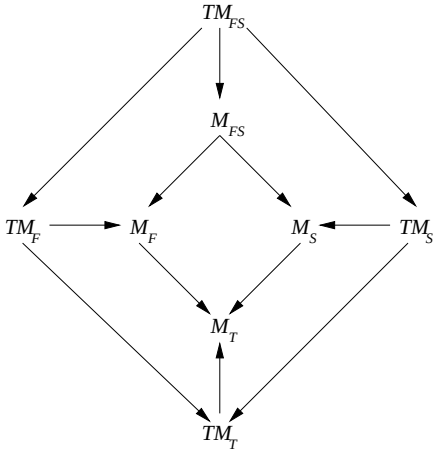

Figure 1. Reed's Hierarchy of Timed and Untimed CSP models. The suffices *T*, *S* and *F* stand for Traces, Stability and Failures respectively.

2.2. Timed Traces, Refusals, Failures and Stability

There are several new notions that are central to the semantics of Timed CSP: *timed events*, *timed traces*, *timed refusal sets* and *stability values*, which we now describe.

2.2.1. Timed Events and Traces

A *timed trace* is a finite sequence of observable events performed by a process, such that each event is labelled with the time at which it occurs. The events in the sequence are ordered chronologically. There are two forms of timed traces, one for the TM_F and TM_{FS} models, and one for the TM_T and TM_S models.

The set of all observable events is Σ. The time domain is the non-negative reals $TIME = [0, \infty)$. A *timed event* is an ordered pair (t, a), where $a \in \Sigma$ and $t \in TIME$. The set of all timed events is $T\,\Sigma = TIME \times \Sigma$. The set of all timed traces for the TM_F and TM_{FS} models is defined as follows:

$$(T\,\Sigma)^*_{\leq} = \{s \in T\,\Sigma^* \mid \langle (t_1, a_1), (t_2, a_2)\rangle \preccurlyeq s \Rightarrow t_1 \leq t_2\} \qquad \text{(Timed Traces)}$$

where $s_1 \preccurlyeq s_2$ iff s_1 is a subsequence of s_2.

In addition $\Sigma(s)$ is the set of events occurring in s. If $s, w \in (T\,\Sigma)^*_{\leq}$ then define $s \cong w$ if and only if s and w are the same, *except* that the order of events at the same time may differ.

In both of the TM_T and TM_S models it is necessary to record when an event is performed as soon as it becomes available. This is required to provide a correct treatment of hiding ($\backslash$) in these models, because a hidden event is required to occur as soon as is possible. (For the other timed models TM_F and TM_{FS} this information can be inferred from the timed refusals associated with the trace.) This information is indicated in a trace for these models by denoting the event with a circumflex, e.g. $\hat{a}$. This set of events is denoted by: $\hat{\Sigma} = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$. This is then used to define timed traces for these two models as follows:

$$(T\,\hat{\Sigma})^*_{\leq} = \{s \in T\,\hat{\Sigma}^* \mid \langle (t_1, a_1), (t_2, a_2)\rangle \preccurlyeq s \Rightarrow t_1 \leq t_2\} \qquad \text{(Timed Traces)}$$

In these models, it is also the case that when an $\hat{a}$ event occurs, it can also occur normally as simply an a event. To facilitate this, the function $\tilde{\mathrm{s}}$ is defined over these types of traces and simply removes the circumflexes from all of the $\hat{a}$ events in s.

To illustrate this point consider the process $WAIT\,1; (a \rightarrow P)$. For this process, if a occurs at time 1, then it has occurred as soon as it becomes available and this information is captured by the trace $\langle (1, \hat{a})\rangle$; it also results in the trace $\langle (1, a)\rangle$. However, if a occurs after time 1, say time 2, then it has not occurred when first available, so results in traces $\langle (1, \hat{a})\rangle$ and $\langle (2, a)\rangle$.

2.2.2. Timed Refusals and Failures

In the untimed CSP models that use *failures*, (s, X), to denote processes, the refusal set X represents the set of events that may be refused after the process has performed the trace s. In the timed models TM_F and TM_{FS} a *timed failure* not only represents what a process may refuse after performing a timed trace, but can also include a record of the refusals as the trace is being performed – for example, before the first event is performed, during the time between consecutive events or after the final event of the trace. Consequently, a timed *refusal set* is a (finite) union of "initial", "intermediate" and "final" sets of timed *refusal tokens* that each describe refusal information (with timings) at various stages during the execution of the associated timed trace.

An additional factor (introduced by Reed and Roscoe [9,11]) that affects the definition of timed refusal sets is the "realism" requirement: a process can only change state a finite number of times during a finite time period. This is known as *finite variability*. The effect of this requirement on timed refusal sets is that they can only be formed from a finite number of refusal tokens, as each one corresponds to a state change.

A *timed failure* is then straightforwardly defined as a timed trace combined with a timed refusal. This represents an observation of a process performing the timed trace while refusing sets of events during the time intervals described by the timed refusal.

These informal notions are captured by the following formal definitions:

$$I : TINT = \{\, [l(I),\, r(I)) \mid 0 \leq l(I) < r(I) < \infty \,\} \qquad \text{(Time Intervals)}$$

$$RT : RTOK = \{\, I \times X \mid I \in TINT \wedge X \in \mathbb{P}(\Sigma) \,\} \qquad \text{(Refusal Tokens)}$$

$$\aleph : RSET = \{\, \bigcup Z \mid Z \subseteq RTOK \wedge Z\,finite \,\} \qquad \text{(Refusal Sets)}$$

$$(s,\, \aleph) : TFAIL = \{\, (s,\, \aleph) \mid s \in (T\,\Sigma)^*_{\leq} \wedge \aleph \in RSET \,\} \qquad \text{(Time Failures)}$$

A *time interval* is a finite half-open time interval. The set of all time intervals is $TINT$[4]. A *refusal token RT* is a set product, $I \times X$, where I is a time interval and X is a set of events that

[4]In this definition $r(I)$ and $l(I)$ are the projection functions that return the two values for a time interval I.

can be refused *continuously* during the time interval. The set of all refusal tokens is *RTOK*. A *timed refusal* $\mathcal{N}$ is a finite union of refusal tokens. The set of all timed refusals is *RSET*. In summary, a *timed refusal* represents the times at which events may be refused during the execution of a timed trace, e.g. a set of $(time, event)$ pairs. A *timed failure* $(s, \aleph)$ is a pair composed of a timed trace *s* and a timed refusal $\aleph$.

There are two standard functions (see [11]) defined on timed refusals: $I(\aleph)$ returns the time interval covered by the refusal set $\aleph$ and $\Sigma(\aleph)$ returns the set of events in $\aleph$. These two functions are defined as follows:

$$I(\aleph) = \{\, t \in [0, \infty) \mid \exists a \in \Sigma.(t, a) \in \aleph \,\}$$
$$\Sigma(\aleph) = \{\, a \in \Sigma \mid \exists t \in [0, \infty).(t, a) \in \aleph \,\}$$

2.2.3. Stability

In Timed CSP, *stability* is used to model the internal activity of a process. It is the dual of divergence, as is used in the failure-divergence model of CSP. A process is defined as being *stable* once it has ceased all internal activity. When a process is stable it cannot change state without performing an external event. The *stability value* associated with an observation (timed trace or failure) of a process is the earliest time by which all internal activity of the process is guaranteed to have stopped. It is formally defined to be the least upper bound of all the times when the process becomes stable given the observation. A process which diverges has a stability value of ∞. α is usually used to denote stability values as follows:

$$\alpha : \mathit{TSTAB} \;=\; \mathit{TIME} \cup \{\infty\} \qquad \text{(Stability Values)}$$

Stability values are not used in either the TM_T timed traces or TM_F timed failures models. However, in the TM_S stability model, a stability value is associated with every timed trace. For example, (s, α) where *s* is a timed trace and α is the stability value for the trace *s*, i.e. the time by which internal activity has ceased after performing *s*. Similarly, for the TM_{FS} failure-stability model, a stability value is associated with every timed failure. For example, $(s, \alpha, \aleph)$ where α is the stability value associated with the failure with trace *s* and refusal $\aleph$.

2.3. Termination and Parallel Operators in Timed CSP [5]

We now discuss the different forms of parallel operators used in the different versions of Timed CSP. Although the termination semantics of $|||$ can result in inconsistent processes in untimed CSP (as shown in Section 1), it has been used to advantage by Davies and Schneider [18,17,22] to define event interrupt and timed interrupt (*tireout*) operators in Timed CSP. These operators were originally defined using $|||$ and ";" in such a way to ensure that $\checkmark$s only occur at the end of traces. However, in Schneider's later treatment [27], these operators are defined directly. The problem of inconsistent termination semantics associated with the use of the original version of $|||$ in Timed CSP suggests that this form of $|||$ should not be used in CSP or Timed CSP explicitly, but at best only as a means to define other operators.

The use of a *generalised* or *interface* style parallel operator is now standard within the CSP community, examples in (untimed) CSP can be found in [5,15,27]. A version has also been defined for the timed failures model TM_F for Timed CSP by Davies [18] called the *communicating* parallel operator. However, there are slight differences between the definitions used by various authors. For example, whether $\checkmark$ is an element of the synchronisation set.

In their early work on Timed CSP, Davies and Schneider [26] also use synchronous termination semantics for all their parallel operators in Timed CSP. Their method is similar to

[5] This section contains material extracted from Section 6.4 of [6] by permission of Springer.

the one used by the authors in CSP_T. In particular, they use their generalised parallel operator $P[|X|]Q$ with $X = \{\checkmark\}$ to re-define $|||$. In the definition of $_A\|_B$ they require $\checkmark$ to be implicitly included in A and B. However, they still use the old version of $|||$ to define the event and time interrupt operators and have not added a termination axiom.

3. Factors Influencing the Definition of a Termination Axiom

In this section we begin by summarising the factors and issues the authors considered relevant when defining and selecting a termination axiom for CSP_T. This was done in relation to Brookes and Roscoe's [2] original CSP model N. (For full details see [6]). We shall then consider the factors that we believe should be considered when selecting a termination axiom for Timed CSP, in relation to the Reed and Roscoe Timed CSP models of Figure 1.

3.1. Factors in CSP_T[6]

It was our intention that in CSP_T a $\checkmark$ signifies the successful termination of any (non-divergent) process, whether it is a sequential or parallel process. This means that if a $\checkmark$ occurs at the end of the trace of a (non-divergent) parallel process, then it has *successfully terminated*. Our aim was to ensure that a $\checkmark$ signified not just sequential process termination but also parallel termination, which was not the case in the original CSP. This is captured by the following property on traces:

Definition 1: A process trace satisfies the $\checkmark$*-requirement* if a $\checkmark$ only occurs at the end of the trace.

We extend this definition to processes as follows:

Definition 2: A process satisfies the $\checkmark$*-requirement* if all its traces satisfy the $\checkmark$*-requirement*.

Next, we considered how to capture this intuitive idea by considering four candidate termination axioms. The selected one was added to the existing CSP process axioms (D1) to (N5), see Appendix. The consequence of adding our chosen axiom to the existing process axioms was to exclude from this new failure-divergence model all processes whose traces do not satisfy our $\checkmark$-requirement. The resulting CSP_T model (N_T) is a sub-model of the original CSP failure-divergence model (N), such that, within this new sub-model all processes satisfy the $\checkmark$-requirement.

In our search to define an axiom that captures our notion of successful termination, it became clear that we needed to decide what types of processes the axiom should be applied to. Thus, we needed to resolve the following questions.

- Should the $\checkmark$-requirement apply only to those processes which can never diverge (i.e. those with no divergent traces) or to all processes?
- If it does apply to all processes should it apply to all traces of the process or only to the non-divergent ones?

It is necessary to consider the $\checkmark$-requirement in relation to divergence because in the original CSP model N (see [2]), a diverging process can perform any and every trace; in particular it can perform traces which do not satisfy the $\checkmark$-requirement. Hence, it is necessary to decide to which processes and traces the $\checkmark$-requirement should be applied. We considered three main types of termination axioms:

[6]This section contains material extracted from Sections 4.1 and 4.2 of [6] and Section 3.1 of [7] by permission of Springer and Elsevier, respectively.

1. those that apply only to non-divergent processes;
2. those that apply to divergent and non-divergent processes;
3. those that apply only to the non-divergent traces of both divergent and non-divergent processes.

From these three cases we formalised the following four termination axioms: (TA1) corresponds to the first case, (TA2) and (TA3) are alternatives corresponding to the third case and (TA4) corresponds to the second case.

$$t \neq \langle\,\rangle \wedge D = \varnothing \wedge (s^\frown\langle\checkmark\rangle, \varnothing) \in F \;\Rightarrow\; (s^\frown\langle\checkmark\rangle^\frown t, \varnothing) \notin F \tag{TA1}$$

If the divergent set of a process is empty and $(s^\frown\langle\checkmark\rangle, \varnothing) \in F$ (i.e. $s^\frown\langle\checkmark\rangle$ is a trace) then $s^\frown\langle\checkmark\rangle^\frown t$ is not a trace for any non-null extension t (i.e. no more events can be performed after a $\checkmark$ has occurred). However, if divergence is possible (i.e. $D \neq \varnothing$), there are no constraints on what can happen after a $\checkmark$ (even for non-divergent traces).

$$t \neq \langle\,\rangle \;\wedge\; (s^\frown\langle\checkmark\rangle^\frown t, \varnothing) \in F \;\Rightarrow\; s^\frown\langle\checkmark\rangle \in D \tag{TA2}$$

If a process can perform the trace $s^\frown\langle\checkmark\rangle^\frown t$, where $t \neq \langle\,\rangle$, then the process must have started diverging no later than after performing the $\checkmark$.

$$t \neq \langle\,\rangle \;\wedge\; (s^\frown\langle\checkmark\rangle^\frown t, \varnothing) \in F \;\Rightarrow\; s \in D \tag{TA3}$$

If a process can perform the trace $s^\frown\langle\checkmark\rangle^\frown t$, where $t \neq \langle\,\rangle$, then the process must have started diverging no later than after performing the s (i.e. before the $\checkmark$). A consequence of this is that after such a process has performed s, no matter what (if any) further events it performs, it can always refuse any set of events.

$$t \neq \langle\,\rangle \wedge (s^\frown\langle\checkmark\rangle, \varnothing) \in F \;\Rightarrow\; (s^\frown\langle\checkmark\rangle^\frown t, \varnothing) \notin F \tag{TA4}$$

If $s^\frown\langle\checkmark\rangle$ is any trace, then the process cannot perform any further events after the $\checkmark$. Intuitively this means that once a $\checkmark$ has been performed the process stops, even if it was diverging at the time.

3.1.1. Comparison of the Four Termination Axioms

Having defined our four candidate termination axioms, we investigated their potential effect on various forms of both divergent and non-divergent parallel processes.

The first axiom, (TA1) reflects the view that if it is possible for a process to diverge then we do not expect it to be well-behaved with respect to successful termination. (TA2) and (TA3) capture the idea that if a process can diverge after some trace it is still required to be well-behaved with respect to successful termination as long as it is not actually diverging. However, when it is diverging it is allowed to *misbehave* with respect to successful termination. (TA4) captures the view that a process is still required to be well-behaved with respect to successful termination even when it is diverging.

Reflecting on these, (TA4) is immediately incompatible with Hoare's original axioms since it excludes divergent processes and we can abandon it. (Recall that a guiding principal is that we want to minimise any change to the original semantics.) However, (TA1), (TA2) and (TA3) are all compatible. One possible reason for adopting (TA1) is that it captures the view that if a process can diverge then we do not care whether it is well-behaved with respect to termination. At first, this seems appealing because we might argue that we do not care whether it violates the $\checkmark$-requirement given that it can diverge. However, the reason for not adopting it is that it is too weak to enforce the $\checkmark$-requirement over all processes. We therefore reject (TA1). However, this does not happen with (TA2) or (TA3) and they are both suitable for the following reasons.

- They enforce the ✓-requirement on all non-divergent traces of a process irrespective of whether it diverges or not, which (TA1) does not. This view seems to capture more closely our intuitive ideas which originally lead to the ✓-requirement and the search for a termination axiom.
- If a process can perform a trace of the form $s^\frown\langle\checkmark\rangle^\frown t$, where $t \neq \langle\,\rangle$, then we now know that it is diverging; whereas without this axiom this was not the case. So the only processes which can now exhibit the undesirable behaviour of not satisfying the ✓-requirement are those which do so because they are diverging, which is consistent with their interpretation as the most chaotic process.

The difference between (TA2) and (TA3) is at what point the process must have started diverging to produce the undesired trace. In (TA2) the process must have started no later than after the ✓ whereas in (TA3) it must have started before the ✓. In this sense we view (TA3) as being stronger than (TA2), and in fact we can prove (see [13]) that by adopting (TA3) in favour of (TA2), that all processes which are equivalent to, or contain as a subprocess the following process, would be excluded:

$$\checkmark \rightarrow \bot$$

So our final choice depends on whether we wish to eliminate processes of the form $\checkmark \rightarrow \bot$. The intuitive meaning of this process is that it first successfully terminates then diverges, which is at least intuitively contradictory, if nothing else. Therefore, on this basis, we choose to adopt the termination axiom (TA3) in favour of (TA2).

To summarise, the main reasons for choosing (TA3) (renamed T1) was that it solved the termination problems[7], most closely captured our intuitive views of termination and achieved this with only a minimal modification to the original semantics. In particular, (T1) reflects the view that even if a process can diverge after some trace it is still required to be well-behaved with respect to successful termination. That is to say it satisfies the ✓-requirement when it is not actually diverging; but when it is diverging it is allowed to *misbehave* with respect to successful termination. In this respect, (T1) also maintains the original view of Hoare, Brookes and Roscoe that a diverging process cannot recover by performing a ✓.

This concludes the outline of the process by which the authors selected a termination axiom for an untimed version of CSP, i.e. CSP_T. Once the (T1) axiom was chosen, it was necessary to prove that all of the processes and operators of CSP_T satisfied it and that this new model N_T was consistent and well-defined – see [6] for details. In the next section we discuss the issues that need to be considered in selecting termination axioms for each of the timed models for Timed CSP.

3.2. Factors in Timed CSP

Clearly the new notions of timed traces, timed refusals and stability values that are central to Timed CSP, need to be taken into account when defining suitable termination axioms to add to the various timed models. Of these, probably the most significant is the use of a stability value (the time at which a process ceases internal activity, i.e. becomes "stable") to model divergence, as compared to a divergence trace (a trace after which infinite internal activity is possible) used in CSP.

In addition, in the timed models that include stability values (Timed Stability TM_S and Timed Failures-Stability TM_{FS}), there appears to be an implicit notion of *"immediate stability at termination"*. That is the stability value for a process that performs a ✓ is the same as the time at which the ✓ was performed. This can be inferred from the stability values associated with traces that end in ✓, as defined in the semantic functions for *SKIP*, $WAIT\,t$

[7]For example, the intuitively contradictory processes described in Section 1 are eliminated.

and ";" (see [11]). This contrasts with the stability values defined for other traces, as defined in the semantic function for $\rightarrow$, where the delay constant "δ", associated with non-$\checkmark$ events, is added to the time of the event to produce the stability value. Thus, any termination axiom that is to be introduced must be consistent with these additional requirements as well as our earlier $\checkmark$-requirement.

Another constraint on the introduction of a termination axiom relates to the addition of various *timeout* and *interrupt* operators added to Timed CSP by Davies and Schneider [18, 17,22]. The definition of these operators [17] relies on the use of $|||$, thus the introduction of a termination axiom into the timed models would appear to prohibit the definition of these operators. However, it is the *race* termination semantics of $|||$ that is used to define these additional operators. Thus, if the race termination semantics version $|_{\varnothing}$ of $|||$ was introduced into the language of Timed CSP and used to define the timeout and interrupt operators, no inconsistency would arise.

However, recalling the following definition of $|_{\Theta}$:

$$P|_{\Theta}Q \mathrel{\widehat{=}} (P\|_{\Omega}Q); SKIP$$

It is important to note that this would only be the case in versions of Timed CSP where the *time postulates* [9], concerning the lower bound delay, δ, on the length of the time interval between any two consecutive events in the history of a sequential process, is zero when the first event is $\checkmark$, and that hidden events occur as soon as they become available. In the case of $P|_{\Theta}Q$, this is the $\checkmark$ of the first process in the sequential composition, i.e. $P\|_{\Omega}Q$. These two conditions ensure that the race termination semantics version of $|||$, i.e. $|_{\varnothing}$, will not introduce an additional delay into the timeout and interrupt operators.

It is currently not clear to the authors whether there are other factors, to those given above, which would need to be considered when defining termination axioms for the timed models. However, based on this preliminary analysis we now present *candidate* termination axioms for the four timed models that would be a starting point for any future research aimed at providing a consistent treatment of termination within these Timed CSP models (Section 5).

3.3. Adding Termination Axioms to the Timed Models

In this section, we outline how termination axioms could be added to the process axioms of the timed semantic models from Reed's hierarchy, shown in Figure 1. In the following sections, we present candidate termination axioms for each of them: *Timed Traces* (TM_T), *Timed Stability* (TM_S), *Timed Failures* (TM_F) and *Timed Failures-Stability* (TM_{FS}). Reed's original axioms for these models, which do not consider termination, can be found elsewhere [9,11].

If a termination axiom (TA) were added to a Timed CSP model, then it would be necessary to prove that each new model, for example $(TM_{FS} + TA, d)$, is a well-defined metric space, where the processes satisfying the axioms $TM_{FS} + TA$ form the "set of points" of the metric space. The "metric", d, is a function that measures the "distance" between two points, i.e. processes. The distance metric used by Reed and Roscoe [9,11] is based on the length of time it takes to tell two processes apart, based on the observations made (e.g. timed traces and timed refusals). In essence, if two processes can be distinguished very quickly, then the "distance" between them is large; but if it takes a long time to distinguish between them, then the "distance" is small.

Processes are formally denoted by their observable behaviours, $\mathcal{S}$, where what is observed depends on the semantic model (e.g. $\mathcal{S}$ is a set of timed failures in the TM_F model). These observations, $\mathcal{S}$, are elements in the point set of the metric space, provided they satisfy the axioms of the model under consideration. So, a process P is denoted by a set of observations $\mathcal{S}$ that satisfy the axioms for the particular semantic model.

3.3.1. A Termination Axiom for TM_T

The Timed Trace model TM_T, models processes by denoting them by sets of timed traces s. The termination axiom that we believe is the most suitable candidate for this model is the following:

$$s \in \mathcal{S} \wedge \checkmark \in \Sigma(s) \;\Rightarrow\; \exists\, t \in [0,\, \infty) \bullet \widetilde{s} = s' \frown \langle (t, \checkmark) \rangle \wedge \checkmark \notin \Sigma(s') \qquad (TA_T)$$

The TA_T axiom[8] captures the notion that if a process can perform the timed trace s and a $\checkmark$ has occurred in the trace then:

(i) only one $\checkmark$ can have occurred; and

(ii) it must have been the last event.

In other words, this axiom would then ensure that the $\checkmark$-requirement was enforced on the timed trace s.

3.3.2. A Termination Axiom for TM_S

The Timed Stability model TM_S models processes by denoting them by sets of pairs (s, α) consisting of a timed trace s and an associated stability value α. The termination axiom we believe is the most suitable candidate for this model is the following:

$$(s, \alpha) \in \mathcal{S} \wedge \checkmark \in \Sigma(s) \;\Rightarrow \widetilde{s} = s' \frown \langle (\alpha, \checkmark) \rangle \wedge \checkmark \notin \Sigma(s') \qquad (TA_S)$$

The TA_S axiom captures the notion that if a process can perform the timed trace s with a stability value of α and a $\checkmark$ has occurred in the trace then:

(i) the $\checkmark$-requirement is satisfied; and

(ii) the $\checkmark$ occurred at time α, the time of stability.

This axiom and the trace prefix closure axiom together would ensure that the $\checkmark$-requirement was enforced and that stability on termination was maintained.

3.3.3. A Termination Axiom for TM_F

The Timed Failures model TM_F models processes by denoting them by sets of timed failures $(s, \aleph)$, consisting of a timed trace s and a timed refusal set $\aleph$. The termination axiom we believe is the most suitable candidate is the following:

$$\begin{aligned} & (s, \aleph) \in \mathcal{S} \wedge \checkmark \in \Sigma(s) \qquad (TA_F) \\ \Rightarrow \quad & \exists\, t : [0,\, \infty) \bullet s = s' \frown \langle (t, \checkmark) \rangle \wedge \checkmark \notin \Sigma(s') \wedge (s, \aleph \cup \aleph_1) \in \mathcal{S} \end{aligned}$$

where $I(\aleph_1) \subseteq [t, \infty)$. The TA_F axiom captures the notion that if a process can perform the timed trace s while refusing $\aleph$ and a $\checkmark$ has occurred in the trace then:

(i) the $\checkmark$-requirement is satisfied; and

(ii) from the time t at which the $\checkmark$ occurred it can henceforth refuse all further events.

[8] See Section 2.2.1 for the reason why there is an $\widetilde{s}$ in axioms TA_T and TA_S, but not TA_F and TA_{FS}.

3.3.4. A Termination Axiom for TM_{FS}

The Timed Failures-Stability model TM_{FS} models processes by denoting them by sets of timed failures paired with associated stabilities. For simplicity, we flatten these pairs into triples $(s, \alpha, \aleph)$, consisting of a timed trace s, a stability value α and a timed refusal set $\aleph$. We believe the most suitable termination axiom for this model is the following:

$$(s, \alpha, \aleph) \in \mathcal{S} \wedge \checkmark \in \Sigma(s) \qquad (TA_{FS})$$
$$\Rightarrow \quad s = s' \frown \langle(\alpha, \checkmark)\rangle \wedge \checkmark \notin \Sigma(s') \wedge (s, \alpha, \aleph \cup \aleph_1) \in \mathcal{S}$$

where $I(\aleph_1) \subseteq [\alpha, \infty)$.

The TA_{FS} axiom captures the notion that if a process can perform the timed trace s with a stability value of α while refusing $\aleph$ and a $\checkmark$ has occurred in the trace then:

(i) the $\checkmark$-requirement is satisfied;
(ii) the $\checkmark$ occurred at the time of stability α; and
(ii) that from time α it can henceforth refuse all further events.

4. Related Work

The authors are only aware of these particular issues regarding the modelling of successful termination within Timed CSP, having been addressed by the work of Schneider [27]. In [27] Schneider, presents an updated version of Reed and Roscoe's original Timed-Failures model [9], but does not consider any of the other timed models.

Schneider's Timed-Failures model has removed the use of the after action delay constant "δ"; all actions are instantaneous and there is no lower bound on the time between sequential actions, as there was in the original models of Reed and Roscoe [8,9,10]. In addition, an equivalent $\checkmark$-requirement has also been introduced, captured in the definition of the set of timed traces *TT* (see page 336 of [27]), i.e. that if a $\checkmark$ occurs in a timed trace then it must be the final event, even if other events occur at the same time. Together, these modifications result in an improved treatment of termination, within the Timed-Failures model.

However, all of the parallel operators Schneider defines in this version of Timed CSP: *alphabetised* $({}_A\|_B)$, *interleaving* $(|||)$ and *interface* (generalised) $(\|_A)$ use synchronous termination semantics. Therefore, this version of Timed CSP could be augmented by the introduction of timed versions of our asynchronous $(|||_\Theta)$ and race $(|_\Theta)$ parallel operators, and thus, provide a more flexible language for designing parallel systems.

Currently, the authors are not aware of any other modification of the models for Timed CSP developed by Reed and Roscoe [9,8,11], by either the addition of any termination axioms or by the modification to the existing axioms, in order to capture a more consistent use of $\checkmark$ in these models.

5. Conclusions and Further Work

We began this paper by outlining how the authors defined an improved model for untimed CSP, with respect to termination, called CSP_T. This was achieved by adding a termination axiom to the original failure-divergence model for CSP and replacing the original parallel operators with more flexible ones. Based on our experience of having successfully achieved this process for untimed CSP, we outlined the various issues that determined our choice of a termination axiom that best captured our notion of successful termination within the context of an untimed model.

Our goal in this paper was to achieve the same outcome for Timed CSP. Thus, we decided to adopt a similar approach for defining alternative models for Timed CSP. To this end, we have proposed a collection of termination axioms for several of the models for Timed CSP. As with untimed CSP, we have considered several factors that influence the choice and definition of a termination axiom within the family of Timed CSP models, the most significant of these is the notion of stability. In particular, when should a process become stable on termination: either at termination or at some time afterwards? In the axioms we have proposed, we have chosen to adopt the former, which we believe is the most appropriate, within the Timed CSP world.

As noted in Section 3.3, the next step in this work is formally to add each axiom to its respective model, by proving that each new model, for example $(\mathit{TM}_{FS} + \mathit{TA}, d)$, represents a complete, bounded, ultra-metric space and further, that the operators are all well-defined in the new model. The authors assume that these proofs would be similar to those presented by Reed and Roscoe [9,11] for the existing axioms of TM_{FS}.

In analysing the timed versions of CSP our intention is to lay the ground work for the definition of timed equivalents of our three untimed parallel operators (synchronous, asynchronous and race) that were introduced in CSP_T. To be able to achieve this goal, it will first be necessary to ensure that successful termination is well-defined within the models of Timed CSP, which we believe we have begun by the definition of these candidate termination axioms.

Acknowledgements

We are extremely indebted to the anonymous reviewers of this paper who provided very valuable feedback and helped us to improve the readability of the paper. Thank you. We would like to gratefully acknowledge the advice and assistance of the following individuals from whom at various stages we benefited a great deal over several very detailed discussions on our work: Jonathan Bowen, Michael Luck and Steve Schneider. We would also like to acknowledge both Springer and Elsevier for granting us permission to use extracts from our previous papers. Finally, we would like to acknowledge the help and assistance of the CPA 2013 Editors, especially Peter Welch, in preparing this paper.

References

[1] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(7), 1985.

[2] S. D. Brookes and A. W. Roscoe. An improved failures model for Communicating Sequential Processes. In *Proceedings of Pittsburgh Seminar on Concurrency*, LNCS 197, pages 281–305. Springer-Verlag, 1985.

[3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985. ISBN: 0-131-53271-5.

[4] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1997. ISBN: 0-13-674409-5.

[5] A.W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010. ISBN: 978-1-84882-257-3.

[6] P. Howells and M. d'Inverno. A CSP model with flexible parallel termination semantics. *Formal Aspects of Computing*, 21(5):421–449, 2009.

[7] P. Howells and M. d'Inverno. Specifying Termination in CSP. *Theoretical Computer Science*, 2013. DOI: 10.1016/j.tcs.2013.05.008.

[8] G.M. Reed and A.W. Roscoe. A Timed Model for Communicating Sequential Processes. In *Proceedings of ICALP'86*, LNCS 226, pages 314–323. Springer-Verlag, 1986. (Also, Theoretical Computer Science, 58, pages 249–261, 1988.).

[9] G.M. Reed. *A Uniform Mathematical Theory of Distributed Computing*. PhD thesis, Oxford University, 1988.

[10] G.M. Reed and A.W. Roscoe. Analyzing TM_{FS}: A Study of Nondeterminism in Real-Time Concurrency. In *Proceedings of 2nd UK-Japan CS Workshop*, LNCS 491, pages 36–63. Springer-Verlag, 1991.
[11] G.M. Reed and A.W. Roscoe. The timed failures - Stability model for CSP. *Theoretical Computer Science*, 211(12):85 – 127, 1999.
[12] P. Howells and Mark d'Inverno. Specifying and Analysing Networks of Processes in CSP_T (or In Search of Associativity). In *Proceedings of Communicating Process Architectures 2013 (CPA13)*, 2013.
[13] P. Howells. *Communicating Sequential Processes with Flexible Parallel Termination Semantics*. PhD thesis, University of Westminster, 2005.
[14] S. D. Brookes. *A Model for Communicating Sequential Processes*. PhD thesis, Oxford University, 1983.
[15] H. Tej and B. Wolff. A Corrected Failure-Divergence Model for CSP in Isabelle/HOL. In *Proceedings of the FME '97 – Industrial Applications and Strengthened Foundations of Formal Methods*, LNCS 1313. Springer-Verlag, 1997.
[16] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.
[17] J. Davies and S. Schneider. An Introduction to Timed CSP. Technical Monograph PRG-75, Programming Research Group, Oxford University, 1989. Available from `https://www.cs.ox.ac.uk/files/3399/PRG75.pdf`.
[18] J. Davies. *Specification and Proof in Real-Time Systems*. Cambridge University Press, 1993.
[19] J. Davies, D. Jackson, and S. Schneider. Broadcast Communication for Real-time Processes. In *Proceedings of Symposium on Real-time and Fault-tolerant Systems*, LNCS 571, pages 149–169. Springer-Verlag, 1991.
[20] J. Davies and S. Schneider. Factorizing Proofs in Timed CSP. Technical Monograph PRG-75, Programming Research Group, Oxford University, 1989. (Also, LNCS 442, pages 129–159, Springer-Verlag, 1990).
[21] J. Davies and S. Schneider. Using CSP to Verify a Timed Protocol over a Fair Medium. In *Proceedings of CONCUR'92*, LNCS 630, pages 355–369. Springer-Verlag, 1992.
[22] S. Schneider. Correctness and Communication in Real-time Systems. Technical Monograph PRG-84, Programming Research Group, Oxford University, 1990. Available from `https://www.cs.ox.ac.uk/files/3408/PRG84.pdf`.
[23] S. Schneider. Timewise Refinement for Communicating Processes. In *Proceedings of 9th Workshop on Mathematical Foundations of Programming Language Semantics*, LNCS 802, pages 162–193. Springer-Verlag, 1993.
[24] S. Schneider. An Operational Semantics for Timed CSP. *Information and Computation*, 116(2):193–213, 1995.
[25] J. Davies, D.M. Jackson, G.M. Reed, J.N. Reed, A.W. Roscoe, and S. Schneider. Timed CSP: Theory and Practice. In *Proceedings of REX Workshop on Real-Time: Theory in Practice*, LNCS 600, pages 640–675. Springer-Verlag, 1992.
[26] J. Davies and S. Schneider. A Brief History of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995.
[27] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 2000. ISBN: 0-471-62373-3.
[28] G.M. Reed. A Hierarchy of Domains for Real-time Distributed Computing. In *Proceedings of 5th Workshop on Mathematical Foundations of Programming Language Semantics*, LNCS 442, pages 80–128. Springer-Verlag, 1990.
[29] G.M. Reed and A.W. Roscoe. Metric-spaces as Models for Real-time Concurrency. In *Proceedings of 3rd Workshop on Mathematical Foundations of Programming Language Semantics*, LNCS 298, pages 331–343. Springer-Verlag, 1987.
[30] W.A. Sutherland. *Introduction to Metric and Topological Spaces*. Oxford University Press, 1981.

Appendix: CSP_T Semantic Definitions

This appendix contains a summary of the notation and definitions used in the paper; for a more detailed description of CSP see [3,4,5] and for CSP_T see [6,7].

The notation used in CSP and CSP_T is the following. Σ is the set of all events, and is countable; denoted by a, b, c. $\mathbb{P}(\Sigma)$ is the power set of Σ; denoted by X, Y, Z. $\mathbb{F}(X)$ is the set of finite subsets of X. Σ^* is the set of finite sequences of events, i.e. *traces*; denoted by r, s, t, u. $\langle\,\rangle$ represents the empty sequence. $\langle a, b, c, d\rangle$ represents the sequence with members a, b, c, d. $s^\frown t$ represents the concatenation of the two sequences s and t. $s \leq t$ is the sequence *prefix* relation and $s < t$ is the proper *prefix* relation. $\#s$ is the length of the sequence s.

a in s is sequence membership. $\overline{X}$ represents the set complement of X, with respect to Σ, i.e. $\overline{X} \cap X = \varnothing$ and $\overline{X} \cup X = \Sigma$. $X - Y$ is the set difference of X and Y, i.e. $X - Y = X \cap \overline{Y}$. Processes are denoted by P, Q, R and A, B, C denote their alphabets. The set of all process identifiers (used to define recursive processes) is denoted by Ide, and we use $p, q, \ldots$ to range over Ide.

In the failure-divergence models for CSP and CSP_T a process P is denoted by an ordered pair $\langle F, D \rangle$, where F is the set of *failures* of P and D is the set of *divergence traces* of P. A *failure* is an ordered pair (s, X), where s is a trace and X is a set of events called a *refusal set*. If the failure (s, X) is in the set of failures for P then P can perform the trace s and then may refuse to participate in any of the events in X. A *divergence trace* s is one that P can perform but after it has done so, P may then be *diverging*, i.e. performing an unbounded sequence of internal events.

The formal definition of the traces and alphabet (set of events that it can perform) for a process $P = \langle F, D \rangle$ are defined as follows:

$$\mathsf{traces}(P) = \{\, s \mid \exists X \in \mathbb{P}(\Sigma) : (s, X) \in F \,\}$$
$$\alpha(P) = \{a \mid \exists\, t \in \mathsf{traces}(P) : a \textit{ in } t\,\}$$

The failure set F and divergence set D for a process are then any sets that satisfy the following conditions:

$$F \subseteq \Sigma^* \times \mathbb{P}(\Sigma) \qquad\qquad D \subseteq \Sigma^*$$

In addition, they must also satisfy the following standard set of *process axioms* for the original model of CSP, see [2], as well as our CSP_T *Termination* axiom (T1):

$$s \in D \Rightarrow s^\frown t \in D \tag{D1}$$

$$s \in D \Rightarrow (s^\frown t, X) \in F \tag{D2}$$

$$(\langle\,\rangle, \varnothing) \in F \tag{N1}$$

$$(s^\frown t, \varnothing) \in F \Rightarrow (s, \varnothing) \in F \tag{N2}$$

$$(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F \tag{N3}$$

$$(s, X) \in F \wedge (\forall\, c \in Y : (s^\frown \langle c \rangle, \varnothing) \notin F) \Rightarrow (s, X \cup Y) \in F \tag{N4}$$

$$(\forall\, Y \in \mathbb{F}(X) : (s, Y) \in F) \Rightarrow (s, X) \in F \tag{N5}$$

$$t \neq \langle\,\rangle \wedge (s^\frown \langle \checkmark \rangle^\frown t, \varnothing) \in F \;\Rightarrow\; s \in D \tag{T1}$$

A natural language interpretation of these axioms is as follows:

(D1) states that the divergence set of a process is suffix closed. This captures the idea that once a process has started to diverge it does so for ever and that it is impossible for the process to recover, i.e. stop diverging, by perform some event, even ✓.

(D2) implies that if a process is diverging then it may also fail, i.e. it may refuse any set of events offered to it at any later stage. This captures the totally nondeterministic and chaotic nature of a diverging process in that it is seen as being *catastrophic*. This axiom enforces the consistency requirement between the divergence set and the failure set of a process.

(N1) and (N2) together imply that the traces of a process form a non-empty prefix closed set, i.e. the traces of a process form a *tree*.

(N3) if a process can refuse a set of events X then it can refuse all the subsets of X. If a process is unable to perform any of the events in X then it could not perform a subset of them.

(N4) if at some point a process can refuse the set of events X and there is another set of events Y that it also can can refuse at that point then clearly the process can refuse both of them together, i.e. $X \cup Y$. Basically this means that if it is impossible for a process to perform an event at some point then it can be added to the refusal set at that point.

(N5) means that if a process can refuse all of the finite subsets Y of a (possibly infinite) set X then it can also refuse the set X. This is a *closure* property for refusal sets which allows us to deduce that infinite sets are refusable if all of their finite subsets are refusable.

(T1) means that if a process indicates that it has terminated (by means of the $\checkmark$) but continues to perform events (t), then it must have started diverging before it performed the $\checkmark$ (i.e. $s \in D$). See Sections 1 and 3.1.

Finally, there is a *refinement ordering* ($\sqsubseteq$) defined on CSP processes in terms of the reverse subset ordering on the failure and divergence set components of processes. This ordering then induces an *equivalence* relation ($\equiv$) on processes where: two processes are equivalent in the model N_T when their failure and divergent set components are equal. This space of processes together with the ordering, i.e. ($PROC_T$, $\sqsubseteq$) is a *partially ordered set*, with least element $\perp$. The semantic functions for CSP_T can be found in [6,7].