A CSP model with flexible parallel termination semantics

Paul Howells, Mark d'Inverno

DOI 10.1007/s00165-008-0098-z
BCS © 2008
Formal Aspects of Computing (2009) 21: 421–449

Formal Aspects of Computing

A CSP model with flexible parallel termination semantics

Paul Howells[1] and Mark d'Inverno[2]

[1] School of Informatics, University of Westminster, 115 New Cavendish St., London, W1W 6RU, UK.
Email: P.Howells@wmin.ac.uk
[2] Department of Computing, Goldsmiths, University of London, London SE14 6NW, UK.
Email: dinverno@gold.ac.uk

Abstract. In the original *failure-divergence* semantic model for *Communicating Sequential Processes* (CSP), the incomplete treatment of successful process termination, and in particular parallel termination, permitted *unnatural* processes to be defined. In response to these problems, a number of different solutions have been proposed by various authors since the original failure-divergence model was developed by Hoare, Brookes and Roscoe. This paper presents an alternative solution to this problem, which is both closer to the original semantic model and provides greater flexibility over the type of parallel termination semantics available in CSP.

Keywords: Concurrency; CSP; Termination

1. Introduction

The aim of this paper is to provide an improved and flexible treatment of successful termination in the language and failure-divergence semantic model of Communicating Sequential Processes (CSP), as presented in [BR85, Hoa85]. By improved we mean solving the problem of the mismatch between the intuitive meaning of the notion of the *successful termination* of a process and how this notion is represented in the failure-divergence semantic model. And by flexible we mean the introduction of different types of parallel termination semantics that can be used in the language of CSP.

There is a need to improve the semantic treatment of successful termination in the original model as we shall illustrate presently. The need for greater flexibility of the type of parallel termination semantics available in CSP is due to a desire to be able to model and design parallel systems in different environments. For example, in a distributed system it may be impractical or inefficient to synchronize the termination of processes, whereas asynchronous termination would be more appropriate. However, in a multiprocessor machine synchronous termination would be appropriate. In other environments, for example competing information searches on the web, it may be acceptable to only require one of a number of processes to terminate, rather than insist on all processes terminating. We believe that by providing parallel operators with different termination semantics, it will provide a more natural way of modeling and designing these types of systems.

Correspondence and offprint requests to: P. Howells, Email: P.Howells@wmin.ac.uk

We intend to achieve our overall aim of providing an improved and flexible treatment of the successful termination in CSP, in three main stages:

- modify the semantic model for CSP by the addition of a new process axiom, which captures our notion of successful termination of a process; and
- show that the non-parallel features of the language (with minor restrictions) satisfy the new termination axiom; and
- finally define new parallel operators with different termination semantics to replace the existing ones, ensuring that they satisfy the new termination axiom.

We believe a satisfactory solution can be provided which is both closer to the original semantics of CSP and provides more flexibility in the choice of parallel termination semantics available than is provided by the existing solutions.

1.1. Successful termination of sequential processes in CSP

We provide a brief introduction to CSP in the Appendix. Successful termination of sequential processes is modelled in CSP by means of a special event ✓ (pronounced *tick*). If ✓ occurs at the end of a trace of a process then the process is considered to have successfully terminated. ✓ is used in the definition of the sequential composition of two processes $P; Q$, where if a ✓ occurs at the end of a trace of P it is used to *signal* to Q that P has successfully terminated and hence that Q can start to execute. For example, the following condition holds:

$$s^\frown\langle\checkmark\rangle \in \mathsf{traces}(P) \wedge t \in \mathsf{traces}(Q) \;\Rightarrow\; s^\frown t \in \mathsf{traces}(P; Q)$$

The ✓ at the end of the trace $s^\frown\langle\checkmark\rangle$ of P signals to Q that it can start to execute, in this case the trace t. Note also that the ✓ at the end of $s^\frown\langle\checkmark\rangle$ does not appear in the resultant trace $s^\frown t$ of $P; Q$. The ✓ is *hidden* from the environment of the sequential process and consequently it does not appear in the trace. ✓ is only generated by the process *SKIP*, which successfully terminates and then does nothing else, it is defined as follows:

Definition 1.1

$$SKIP \;\widehat{=}\; \checkmark \rightarrow STOP$$

This illustrates the difference between the two processes *SKIP* and *STOP*, in that *SKIP* successfully terminates then does nothing, whereas *STOP* does not successfully terminates but just does nothing. Consequently when *SKIP* is sequentially composed with another process P, the resultant process is equivalent to P, i.e., *SKIP* is the (left) identity process for ";", and is captured in the following law:

$$SKIP; P \;\equiv\; P \tag{1}$$

Whereas when *STOP* is sequentially composed with another process P, the resultant process is equivalent to *STOP*, i.e., *STOP* is the (left) zero process for ";", this is represented in the following law:

$$STOP; P \;\equiv\; STOP \tag{2}$$

Note that neither of the following equivalences hold in general (see [Hoa85]):

$$P; SKIP \;\equiv\; P \tag{3}$$

$$P; STOP \;\equiv\; STOP \tag{4}$$

In the remainder of the paper, we shall make use of the *standard set* of algebraic laws for CSP, for reasons of brevity we do not include these laws in this paper, they can be found in [Bro83, BR85, Hoa85, How05].

1.2. A known problem with the semantics of CSP

In the *original* [BR85, Hoa85] semantic (trace and failure-divergence set) models for CSP process termination was only dealt with comprehensively for the sequential composition of processes. This was achieved by the interplay of the definition of sequential composition ";" and the use of the special termination event ✓. In these semantic models the termination of parallel processes had only been partially specified, (see [Hoa85, pp. 175–178]). There was no additional special event for the successful termination of the parallel composition of processes, though ✓ is used by default for this purpose. However the definitions of the parallel operators did not take into account the (default) significance of ✓ in representing the successful termination of parallel process, unlike in the definition of ";" for sequential processes. This omission can, under certain circumstances, result in counter-intuitive occurrences of a ✓ in the middle of a trace of a parallel process. This is obviously inconsistent with the intuitive meaning ascribed to ✓, which is intended to represent successful termination. To illustrate the problem we present a simple example.[1]

Example 1.1 Using the two processes $P \mathrel{\widehat{=}} a \rightarrow SKIP$ and $Q \mathrel{\widehat{=}} SKIP$, that both successfully terminate, the trace sets for P and Q are as follows:

$$\mathsf{traces}(a \rightarrow SKIP) = \{\langle\,\rangle, \langle a\rangle, \langle a, \checkmark\rangle\}$$
$$\mathsf{traces}(SKIP) = \{\langle\,\rangle, \langle\checkmark\rangle\}$$

Then the traces of the asynchronous parallel composition of these processes, $P|||Q$ is as follows:

$$\mathsf{traces}((a \rightarrow SKIP)|||SKIP) = \{\,\langle\,\rangle, \langle a\rangle, \langle\checkmark\rangle, \langle a, \checkmark\rangle, \langle\checkmark, a\rangle, \langle a, \checkmark, \checkmark\rangle, \langle\checkmark, a, \checkmark\rangle\,\}$$

Note that in three of these traces a ✓ occurs other than at the end, i.e., $\langle\checkmark, a\rangle$, $\langle a, \checkmark, \checkmark\rangle$ and $\langle\checkmark, a, \checkmark\rangle$, are all valid traces. Clearly the ✓s which do not occur at the end of these traces cannot be interpreted as representing the successful termination of the process $(a \rightarrow SKIP)|||SKIP$ since it continues to perform a and ✓ events after a ✓ has been performed.

This example clearly illustrates that the classic semantics for ||| does not take account of the significance of ✓ when combining processes that can successfully terminate, and results in counter intuitive traces. It also illustrates a general problem of ✓s occurring in the middle of a trace, for example, consider a process P with the following trace:

$$s^\frown\langle\checkmark\rangle^\frown t \in \mathsf{traces}(P) \quad [t \neq \langle\,\rangle]$$

Here the ✓ is not hidden from the environment of P as is the case when it occurs at the end of a trace of the first process in a sequential composition. As we have seen this situation can arise when processes are constructed using the asynchronous parallel operator, and at least one of the processes being combined contains the process *SKIP*. It is this occurrence of ✓ in the middle of a trace that indicates that the termination semantics of the parallel operators was unsatisfactory. This is because ✓ is meant to indicate that a process has terminated successfully and hence that it will not continue to perform events after it has performed a ✓. This is obviously not the meaning of the ✓ in the above case where after $s^\frown\langle\checkmark\rangle$ it continues to perform events, i.e., the trace t.

The occurrence of a ✓ in the middle of a trace of a process is inconsistent because taking ✓ to mean termination, the process signals to its environment that it has finished but then continues to perform events. This type of process could be thought of as a *zombie* process since, in effect it dies and then comes back to life. This mismatch between the expected meaning of a program and its actual meaning in the semantic model is clearly undesirable. It seems clear that a desirable (or even essential) characteristic is that the formal meaning of a program should correspond as closely as possible to our expected intuitive meaning.

A number of authors Hoare [Hoa85], Tej and Wolff [TW97], Roscoe [Ros98] and Hoare and He [HJ98] have proposed various solutions to this problem. We shall discuss these solutions in Sect. 2 and compare them to the solution we present in this paper in Sect. 6.

[1] For the full analysis of this example see Example 3.1 in Sect. 3.2.2.

1.3. Overview of paper

The structure of the remainder of the paper is as follows. In Sect. 2, we present an overview of the existing solutions to the termination problem. In Sect. 3, we characterize and analyse the termination semantics of the existing parallel operators in the standard language of CSP. In Sect. 4 a new model for CSP is defined by adding a *termination* axiom to the existing axioms of CSP. In Sect. 5, we define three replacement parallel operators with different termination semantics, that are consistent with the new model. We compare our solution to the problem with that of other authors in Sect. 6. Finally, we present our conclusions and suggest areas for further research.

2. Existing solutions

Several solutions have been proposed for solving the termination problem in CSP. The original solution proposed by Hoare [Hoa85] involved only syntactic restrictions on the use of CSP. Those of Tej and Wolff [TW97], and Roscoe [Ros98] involved the modification of the original failure-divergence semantic model. The fourth takes a very different approach and is the recasting of CSP within Hoare and He's [HJ98] *Unifying Theories of Programming* (UTP) framework. We shall now present an overview of these solutions to the termination problem.

2.1. Hoare's solution

In the solution proposed by Hoare [Hoa85], a constraint is placed on the use of $|||$, in that *SKIP* should never be a subprocess of P or Q. In addition, the mixed operator $P_{A}\|_{B} Q$, is considered invalid unless the following condition is satisfied:

$$(A \subseteq B) \lor (B \subseteq A) \lor (\checkmark \in (A \cap B) \cup (\overline{A \cup B}))$$

The result of this with respect to the termination semantics of ${}_{A}\|_{B}$ is as follows. When $A \subseteq B$ if $\checkmark \in A$ then ${}_{A}\|_{B}$ has synchronous termination and requires both processes to terminate; if $\checkmark \in \overline{A} \cap B$ then ${}_{A}\|_{B}$ has race termination and only requires Q to terminate successfully; if $\checkmark \notin B$ then it does not terminate, so is consistent with any form of termination semantics. Similarly for $B \subseteq A$. When $\checkmark \in A \cap B$ then ${}_{A}\|_{B}$ has synchronous termination and requires both processes to terminate. When $\checkmark \in \overline{A \cup B}$ then it does not terminate and is consistent with any form of termination semantics. As can be seen this solution still results in inconsistencies in the termination semantics of ${}_{A}\|_{B}$.

2.2. Tej and Wolff's solution

Tej and Wolff [TW97] discovered a problem with the sequential composition operator ";", while attempting to represent CSP within the Isabelle/HOL theorem prover. As a result of finding this initial termination related problem, they developed a solution for the general termination problems by modifying the version of the failure-divergence model defined in [Ros92]. The main changes they make are summarized below.

- Requiring that if a $\checkmark$ occurs then it can only occur at the end of a trace, this applies to both non-divergent and divergent traces.
- A new failure-divergence model capturing their view of termination and how $\checkmark$ should be handled is defined using the following axioms:

  $$(*) \qquad \forall s, X.(s, X) \in F \;\Rightarrow\; \mathsf{front\text{-}tick\text{-}free}(s)$$

  $$(\mathrm{v}^*) \qquad \forall s, t.s \in D \land \mathsf{tick\text{-}free}(s) \land \mathsf{front\text{-}tick\text{-}free}(t) \;\Rightarrow\; s^\frown t \in D$$

  These new axioms (*) and (v*) (a modified (D1)) together only allow $\checkmark$s to appear as final events in traces.
- Modifying the failure and divergence semantic functions for several processes and operators, in particular $\bot$ and $P; Q$, to ensure consistency with the new model.
- Adopting synchronous termination semantics for all of the parallel operators they define.

2.3. Roscoe's solution

Roscoe [Ros98] incorporates a solution, similar in some respect to that of Tej and Wolff, in his latest axiomatization of the failure-divergence model. The solution of the ✓ problem is not his only motivation for presenting this new model that is now considered the "standard" model, due to its use in the FDR tool [FDR03]. The main features of Roscoe's solution to the termination problem are as follows.

- A different view of termination is taken, for example he states that "... termination is really something that the environment observes rather than controls ...". This leads to a need to "... protect it [✓] from confusion with other events.", resulting in ✓ no longer being a member of Σ the set of (normal) events, but as a (non-delayable by the environment) *signal* event, this leads to the definition of $\Sigma^{\checkmark} = \Sigma \cup \{\checkmark\}$. This differs with the view of ✓ taken by Tej and Wolff and by the authors, which is that of the original view, where $\checkmark \in \Sigma$ and it is an event that the environment can refuse if offered by a process.
- This view of ✓ as a signal, different to normal events in Σ has consequences for how the refusals of a process which can offer to terminate should be defined in his model. In particular, he states that "... no process will ever offer its environment the choice of ✓ or events in Σ." This leads to the requirement that if a process has the trace $s^\frown\langle\checkmark\rangle$, it has the failure (s, Σ), which is captured in a new axiom.
  This is partly motivated by his attempt to control the use of "unnatural" processes like $P \,\Box\, SKIP$, which do not fit well with the "... principle that ✓ is something a process signals to say it *has* terminated." And his wish to make equivalence (3) from Sect. 1.1:

  $$P; SKIP \equiv P \tag{3}$$

  hold in his model, note that this does not hold in the original model, if $P = Q \,\Box\, SKIP$. Roscoe [Ros98, pp. 140–141], states that these forms of process cannot be completely eliminated. So his solution is to introduce the *timeout* (or *sliding choice*) operator $\rhd$ and use it to redefine the meaning of $P \,\Box\, SKIP$, by means of the ($\Box - SKIP\ resolve$) law, as follows:

  $$P \rhd SKIP \;\widehat{=}\; (P \,\Box\, SKIP) \sqcap SKIP$$
  $$(\Box - SKIP\ resolve) \qquad P \,\Box\, SKIP = P \rhd SKIP$$

- As with Tej and Wolff, Roscoe requires that if a ✓ occurs then it is always the final event of a trace, this applies for both non-divergent and divergent traces.
- In contrast to Tej and Wolff, Roscoe [Ros98, pp. 139–140] chooses "distributed" (asynchronous) parallel termination semantics for all of his parallel operators. This decision is based on the preference for waiting for all processes to terminate, rather than actively synchronizing the ✓s and that this fits far better with the notion that ||| has distributed termination; and the desire to achieve implementability in a real distributed system.

These views of termination and how ✓ should be handled in the model are reflected in the modified axioms he defines for the model.

(F1) $\mathsf{traces}_\perp(P) = \{\, t \mid (t, X) \in F \,\}$ is non-empty and prefix closed.

(F2) $(s, X) \in F \wedge Y \subseteq X \;\Rightarrow\; (s, Y) \in F$

(F3) $(s, X) \in F \wedge (\forall a \in Y : s^\frown\langle a\rangle \notin \mathsf{traces}_\perp(P)) \;\Rightarrow\; (s, X \cup Y) \in F$

(F4) $s^\frown\langle\checkmark\rangle \in \mathsf{traces}_\perp(P) \;\Rightarrow\; (s, \Sigma) \in F$

(D1) $s \in D \cap \Sigma^* \wedge t \in \Sigma^{*\checkmark} \;\Rightarrow\; s^\frown t \in D$

(D2) $s \in D \;\Rightarrow\; (s, X) \in F$

(D3) $s^\frown\langle\checkmark\rangle \in D \;\Rightarrow\; s \in D$

The axioms (F1) to (F3) and (D2) are similar to (N1) to (N4) and (D2) respectively and (N5) is no longer necessary. Roscoe states that the new axioms (F4), (D1) and (D3) reflect the special role of ✓ and that he does not wish to distinguish between how processes behave after successful termination. Axiom (F4) means that if a process can terminate then it can refuse to do anything but terminate. Axiom (D1) provides divergence closure, note that ✓ can only occur as a final event. Axiom (D3) means that the only way a trace $s^\frown\langle\checkmark\rangle$ can get into D is if it can diverge before termination, that is s is a divergent trace.

2.4. Hoare and He's "unifying theories of programming"

Hoare and He [HJ98] have developed a *unifying theory of programming* that begins the process of unifying many aspects of the science of programming. UTP links the different styles of programming language semantics; represents the software life-cycle activities of specification, design, programming and implementation; and has been applied to several programming paradigms.

UTP [HJ98] achieves this by using an *alphabetized relational calculus* to represent specifications, designs and programs as relations between an initial observation and a subsequent intermediate or final observation. A particular programming paradigm is then modelled by a set of relations that satisfy particular *healthiness conditions* that capture the characteristics of the paradigm being modelled. Hoare and He [HJ98] represent a number of different programming paradigms and languages within their unifying theory, including CSP. (See also [CW06] for a comparison of UTP and Roscoe's CSP model.)

Hoare and He [HJ98] define CSP processes as a subset of *reactive*[2] processes. The relations used to represent reactive processes use several variables to represent observations of different aspects of a process's behaviour. The unprimed versions of the variables represent the initial observations (i.e., observations of its predecessor) and the primed versions represent observations at an intermediate or terminal state of the process. The variables used and their meanings are as followings. ok is true if the previous process is in a stable state and false if it has diverged; ok' is true if the next observation of the process is of a stable state and false if it is of a divergent state. $wait$ is true if the previous process has not terminated and false if it has; $wait'$ is true if the next observation of the process is of an intermediate state and false if it has terminated. tr represents the trace of events that have occurred before the process stared; tr' represents the trace of all events that have occurred up to the next observation. ref represents the set of events that could be refused at the last observation; ref' represents the refusal set in the next observation. The process's own state variables v and v' are also used. For a more detailed explanation of these variables and how they are used see [HJ98, CW06].

Since CSP processes are a subset of reactive processes, the healthiness conditions for CSP include the three healthiness conditions (R1), (R2) and (R3) for reactive process. Where (R1) means that a process can never undo any action performed previously. (R2) means that a process's behaviour is not affected by events that happen before it starts executing. (R3) means that in sequentially composed processes the second process does not start until the first has successfully terminated. So CSP processes satisfy these and the following CSP specific healthiness conditions:

(CSP1) $P = P \vee (\neg\, ok \wedge tr \leq tr')$

(CSP2) $P = P; J$ [where $J = (ok \Rightarrow ok') \wedge v' = v \wedge wait' = wait \wedge tr' = tr \wedge ref' = ref$]

(CSP3) $P = SKIP; P$

(CSP4) $P = P; SKIP$

(CSP5) $P = P \,|||\, SKIP$

CSP1 means no predictions can be made about a process until it has at least started. CSP2 requires a process is monotonic in ok'. CSP3 states that a process does not depend on the initial value of ref. CSP4 means that the value of ref' is irrelevant after a CSP process terminates. CSP5 states that if a process is deadlocked and refusing some set of events offered by its environment, then it would still be deadlocked if the environment offered it fewer of these events.

Since Hoare and He have (CSP4) as a healthiness condition, they like Roscoe, need to deal with processes of the form $P \,\Box\, SKIP$. They achieve this by adopting Roscoe's redefined version of $P \,\Box\, SKIP$. A consequence of this is that the following inequality [HJ98, p. 210] holds:

L5 $P \,\Box\, SKIP \sqsubseteq SKIP$

which by definition of *refinement* ($\sqsubseteq$) means that the following equality holds:

$$P \,\Box\, SKIP = (P \,\Box\, SKIP) \sqcap SKIP$$

which is the one Roscoe uses in his model.

[2] Reactive processes are those with behaviours that can be observed or altered between their initiation and termination.

In UTP parallelism is modelled using *parallel by merge*. In effect when two processes are combined using a parallel operator, the two processes run and each produces its own independent result, these results are then *merged* using a relation to produce the parallel result. Using this approach different forms of parallel operator are modelled by defining appropriate merge relations, in particular, the parallel operators $|||$ and $_A\|_B$. With respect to their termination both of the merge relations used in their definitions require both processes to terminate successfully for their parallel composition to terminate successfully. Furthermore, UTP uses synchronous termination semantics in the definitions of parallel operators. A significant difference between the failure-divergence models for CSP and the UTP approach is that UTP does not use an event (such as ✓) to represent successful termination, but the variables ok' and $wait'$, if $ok' \wedge \neg wait'$ holds then the process has successfully terminated. An advantage of the UTP approach is that it allows other models of concurrency to be interpreted within the unifying theory that do not use ✓, for example those of ACP [BW90].

2.5. Summary

The original solution by Hoare [Hoa85] is only a partial solution at best and clearly was in need of improvement. Tej and Wolff's [TW97] solution solves the general termination problems, but only provides parallel operators with synchronous termination semantics. Roscoe's [Ros98] solution is clearly the most complete within the failure-divergence approach, but again only provides parallel operators with one form of termination semantics, distributed (asynchronous) termination in contrast to Tej and Wolff. Hoare and He [HJ98] solve the termination problems by using their UTP framework and dispensing with the termination event ✓ and replacing its semantic role with the observation variables ok' and $wait'$. As a consequence, the definition of their versions of the CSP parallel operators use synchronous termination semantics and require both processes to terminate. We believe that the solution to the termination problem we shall present in this paper is both simpler, in the sense that it requires fewer modifications to the original semantics, and provides greater flexibility in the choice of parallel operators than the two main solutions of Roscoe [Ros98] and Hoare and He [HJ98] outlined above.

3. Successful termination of parallel processes

We now discuss how the existing notions of successful termination of sequential process could be extended to deal with parallel processes, by identifying several notions of termination which could be used to characterize parallel processes. This is followed by an analysis of the termination semantics of the three original parallel operators.

3.1. Types of parallel termination semantics

We intend that in our improved semantics for CSP a ✓ will signify the successful termination of any (non-divergent) process whether it is a sequential or parallel process. The consequence of adopting this approach is that, if a ✓ occurs at the end of the trace of a (non-divergent) parallel process we take this to mean that it has *successfully terminated*. To help our analysis we introduce the following property on traces:

Definition 3.1 A process's trace satisfies the ✓*-requirement* if a ✓ only occurs at the end of the trace.

So the issue that must be resolved is when should a tick occur in the trace of a parallel process, i.e., when does a parallel process terminate. Therefore, we need to consider the factors that determine this. Clearly, the termination of a parallel process is dependent on two factors:

- whether the constituent processes can successfully terminate, i.e., a ✓ can occur at the end of their traces; and
- how a particular parallel operator combines the termination of its constituent processes to produce the termination of the whole process.

We now identify several forms of termination semantics for parallel composition which characterize some of the circumstances under which a parallel process will or will not terminate depending on the termination of its constituent processes.

In the following we will use $P /\!\!/ Q$ to denote an unspecified type of parallel composition of the two processes P and Q, since here we are only interested in termination, i.e., how ✓ occurs and not how other events occur. To do this we must look at examples of processes constructed using the parallel operator and sequential composition. The reason for this is that in the existing language termination is only dealt with in the context of sequential composition. We must therefore, look at processes of the following form if we wish to examine their termination semantics:

$$(P /\!\!/ Q); R$$

Here P and Q are composed using some form of parallel composition and then sequentially composed with another process R. This type of process allows us to examine under what circumstances R starts to execute, and by implication under what circumstances $P /\!\!/ Q$ terminates, i.e., when a ✓ occurs in a trace of $P /\!\!/ Q$.

We now define the three types of parallel termination semantics we shall use in the remainder of the paper. For other forms of parallel termination semantics see [How05].

- $P /\!\!/ Q$ has *synchronous* termination semantics if its termination requires the successful termination of both P and Q. In addition, P and Q must synchronize on their termination, that is, they must synchronize on the event ✓. So with synchronous termination $P /\!\!/ Q$ will not terminate if either P or Q or both can fail to terminate, or if it is impossible for them to synchronize on ✓. In terms of our test process $(P /\!\!/ Q); R$, R cannot start to execute until both P and Q have terminated by synchronizing on a ✓.
- $P /\!\!/ Q$ has *asynchronous* termination semantics if its termination requires the successful termination of both P and Q; and P and Q terminated asynchronously, i.e., they do not synchronize on ✓. So $P /\!\!/ Q$ will fail to terminate if either P or Q or both fail to terminate. In terms of our test process $(P /\!\!/ Q); R$, again R cannot start until both P and Q have terminated asynchronously. Unlike synchronous termination where P and Q terminate at the same time, i.e., on the same ✓; here P can terminate before Q or vice versa, i.e., on different ✓s. (Roscoe [Ros98] refers to this type of parallel termination semantics as *distributed termination*.)
- $P /\!\!/ Q$ has *race* termination semantics if its termination requires the successful termination of either P or Q; and P and Q terminated asynchronously. So $P /\!\!/ Q$ only fails to terminate if both P and Q fail to terminate. Unlike synchronous and asynchronous termination where the environment of $P /\!\!/ Q$ observes a single ✓, under *race* termination semantics it can observe all ✓s that are performed. So for example it could observe a ✓ from P and a ✓ from Q if both could terminate. The consequence of this in $(P /\!\!/ Q); R$ is that whichever of the two ✓s R (the environment) observes first it takes as representing the termination of $P /\!\!/ Q$ and hence R proceeds to execute, and whichever of P or Q did not terminate is aborted. Hence with this type of termination semantics $P /\!\!/ Q$ can terminate as soon as either P or Q does so; i.e., they terminate asynchronously. (Roscoe [Ros98, p. 139], chooses to reject this form of parallel termination.)

3.1.1. Applications of the three forms of termination semantics

We have identified these three forms of parallel termination semantics since they seem to us to correspond with the most obvious ways in which the termination of parallel processes could be organized. Both synchronous and asynchronous parallel termination semantics have been widely used and studied within CSP [Hoa85, Ros98]. In particular, Roscoe [Ros98] views asynchronous termination semantics as the most appropriate for distributed systems; whereas Hoare and He [HJ98] consider synchronous termination important for studying shared memory systems.

However, race termination has not been widely studied or used except within Timed CSP [Dav93, DS89, Sch90], where it has been used implicitly to define timeout and tireout operators. Thus, some discussion of its usefulness is appropriate. In general we believe parallel operators with race termination semantics would be most appropriate for organizing competing tasks. Examples of such activities are the parallel searching of large amounts of partitioned data and the use of parallel execution of different algorithms for the same task. In either case it is often not predictable which process would complete the task first, so by using race termination semantics in these situations indicates that one is only interested in which process would complete the task first and the slower "losing" ones are in effect to be ignored and terminated. In practice these kinds of task are often performed by the slave process of an *enslavement* operator, thus we believe that race termination would be useful in the study and construction of these types of systems. In particular we could use our race termination operator ($\backslash\!\backslash_{\Theta}$) to define a version of Roscoe's [Ros98] *enslavement* operator ($P //_{\chi} Q$) that, as he admits is not possible for him to define using his operators, in which only the master process is required to terminate without requiring the slave process to terminate.

This version of the enslavement operator would be defined using the generalized race parallel operator rather than the asynchronous parallel operator as follows:

$$P /\!/_X Q = (P \backslash\backslash_X (Q; STOP)) \backslash X$$

The *STOP* after the slave process Q ensures that it can never terminate and so cannot cause the termination of the combination.

3.2. Termination semantics of existing parallel operators

We now analyse the termination semantics of the three parallel operators pure synchronous $\|$, pure asynchronous (interleaving) $|||$ and mixed (alphabetized) $_A\|_B$, with the intention of determining any mismatch between their actual and expected termination semantics. We choose to analyse all three of these operators even though $\|$ and $|||$ can be thought of as special cases of $_A\|_B$, since they are almost always included as separate operators in any presentation of CSP, especially $|||$. In addition, by considering them we are able to highlight the range of problems that exist with the original treatment of termination and to state *desired* equivalences that represent parallel termination healthiness conditions. Our analysis is based on the type of the ✓ event, i.e., if both, one or neither processes can perform it, and whether it is performed synchronously or asynchronously. To do this, we examine examples of our test process $(P /\!/ Q); R$, replacing the unspecified form of parallel composition by the parallel operator under consideration.

3.2.1. Termination semantics of synchronous parallel composition

The pure synchronous parallel operator has synchronous termination semantics under all possible circumstances; this is its natural termination semantics. It is also the case that a ✓ only occurs at the ends of its traces, provided that this is also the case for its subprocesses, (ignoring divergence.) Therefore, we do not need to modify this operator in any way.

3.2.2. Termination semantics of asynchronous parallel composition

We believe that the *natural* termination semantics with respect to our test process $(P|||Q); R$, should require the termination of both P and Q before R starts to execute. This implies that we require it to have either synchronous or asynchronous termination semantics as characterized earlier.

We now intend to examine the circumstances under which $P|||Q$ terminates with respect to the termination of P and Q, obvious $P|||Q$ does not have synchronous termination semantics because no events are synchronized. To examine the termination semantics of $P|||Q$ under these circumstances we shall look at a simple instance of our test process $(P|||Q); R$.

Example 3.1 Consider the case when ✓ is a common asynchronous event, that is $\checkmark \in \alpha(P)$ and $\checkmark \in \alpha(Q)$, thus both P and Q are capable of successfully terminating. We shall use the (divergent free) processes $P \mathrel{\widehat{=}} a \to SKIP$ and $Q \mathrel{\widehat{=}} SKIP$; we shall not specify the process R, but assume that it does not diverge. The failure sets for P and Q are as follows:

$$\begin{aligned}
\mathcal{F}[\![a \to SKIP]\!]e = \quad & \{(\langle\,\rangle, X) \mid a \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle a\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle a, \checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\} \\
\mathcal{F}[\![SKIP]\!]e \quad = \quad & \{(\langle\,\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle\checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\}
\end{aligned}$$

First we construct the failure set for the parallel composition of P and Q; the divergence set for $P|||Q$ is empty.

$$\begin{aligned}
\mathcal{F}[\![(a \to SKIP)|||SKIP]\!]e = \quad & \{(\langle\,\rangle, X) \mid a \notin X \wedge \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle a\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle a, \checkmark\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle\checkmark\rangle, X) \mid a \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle\checkmark, a\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle\checkmark, a, \checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\} \\
& \cup \{(\langle a, \checkmark, \checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\}
\end{aligned}$$

Note that in three of these failure sets a ✓ occurs other than at the end of a trace, i.e., $\langle \checkmark, a \rangle$, $\langle a, \checkmark, \checkmark \rangle$ and $\langle \checkmark, a, \checkmark \rangle$, are all valid traces. This illustrates the point mentioned earlier about the possibility of ✓s occurring else where than at the end of a trace. Clearly the ✓s which do not occur at the end of these traces cannot be interpreted as representing the successful termination of the process $(a \rightarrow SKIP) ||| SKIP$ since it continues to perform a and ✓ events after a ✓ has been performed.

The following process equivalences are valid, since the processes all have the same failure sets:

$$\begin{aligned}(a \rightarrow SKIP) ||| SKIP &\equiv (a \rightarrow \checkmark \rightarrow \checkmark \rightarrow STOP) \,\Box\, (\checkmark \rightarrow a \rightarrow \checkmark \rightarrow STOP) \\ &\equiv (a \rightarrow \checkmark \rightarrow SKIP) \,\Box\, (\checkmark \rightarrow a \rightarrow SKIP) \qquad \text{[Def. } SKIP\text{]} \end{aligned} \tag{5}$$

This illustrates how undesirable these semantics are for |||, since what at first sight looks like a simple innocuous process is equivalent to one which breaks the (informal) rule about ✓s only occurring inside the definition of the process *SKIP*.

Now returning to our test process $(P ||| Q); R$ the divergence set is empty and the failure set is as follows:

$$\mathcal{F}[\![((a \rightarrow SKIP) ||| SKIP); R]\!]e \;=\; \mathcal{F}[\![R]\!]e \cup \{\, (\langle a \rangle^\frown s, X) \mid (s, X) \in \mathcal{F}[\![R]\!]e \,\}$$

From the first failure set $\mathcal{F}[\![R]\!]e$, we can see that the process R can start to execute without an a having to be performed. It is also the case that the following failure set equality holds:

$$\mathcal{F}[\![((a \rightarrow R) \,\Box\, R) \sqcap R]\!]e \;=\; \mathcal{F}[\![R]\!]e \cup \{\, (\langle a \rangle^\frown s, X) \mid (s, X) \in \mathcal{F}[\![R]\!]e \,\}$$

From the definition of the above failure sets and the definition of $\equiv$, the following process equivalence holds:

$$((a \rightarrow SKIP) ||| SKIP); R \;\equiv\; ((a \rightarrow R) \,\Box\, R) \sqcap R \tag{6}$$

This equivalence indicates that in fact the test process is equivalent to one which is a pure non-deterministic choice between two alternatives. In the right hand side (*rhs*) process of (6) we have a pure non-deterministic choice between the following processes.

R – this alternative corresponds to $(a \rightarrow SKIP) ||| SKIP$ terminating immediately by performing the ✓ of the *SKIP* and in doing so signals to R that it can start to execute. Here the environment of the process cannot influence the choice of whether the process performs an a before R proceeds.

$(a \rightarrow R) \,\Box\, R$ – this is the situation where $(a \rightarrow SKIP) ||| SKIP$ does not terminate immediately and allows the environment to decide whether a is to be performed before R starts to execute.

Equivalence (6) illustrates two important points about the original semantics of |||. Firstly, that ||| has race termination semantics. Secondly, it makes explicit the *hidden* non-determinism that ||| introduces in this context. We believe that this introduction of *hidden* non-determinism into a process which given its natural semantics should not contain any, is clearly an undesirable side effect of the original semantics of |||. In general when combining these three processes in this manner one does not intend that the resultant process should be equivalent to the *rhs* process. However, if the *rhs* process is what is really required then it should be explicitly written as such, and not achieved by using ||| and ";". A more natural semantics for ||| should give rise to the following equivalence for our example:

$$(a \rightarrow SKIP) ||| SKIP \;\equiv\; a \rightarrow SKIP \tag{7}$$

Here *SKIP* acts as the identity process for |||. The identity process for ||| with the existing semantics is the process *STOP*, as illustrated by the following law from [BR85]:

$$P ||| STOP \equiv P \tag{8}$$

Now using the desired equivalence (7) rather than law (8) for ||| in the test process we have the following equivalences:

$$((a \rightarrow SKIP) ||| SKIP); R \;\equiv\; (a \rightarrow SKIP); R \;\equiv\; (a \rightarrow R) \tag{9}$$

This equivalence is what would be expected from an asynchronous operator where both processes were required to terminate either synchronously or asynchronously before the composite process could terminate.

Thus we have shown that the pure asynchronous parallel operator has race termination semantics under these circumstances, and it is easy to demonstrate that it has race termination semantics under all circumstances, see [How05]. This is not its natural termination semantics, which we claimed should be either synchronous or asynchronous. It is also the case that ✓s do not only occur at the ends of its traces but in the middle as well. Therefore, we need to replace this operator with one that has the natural semantics of ✓s only occurring at the end of a trace.

3.2.3. *Termination semantics of mixed parallel composition*

We now illustrate by means of an example that mixed parallel operator does not have its natural termination semantics. In particular, with respect to our test process $(P_{A}\|_{B}Q); R$ it should require the termination of both P and Q before R starts to execute. This implies that we require it to have either synchronous or asynchronous termination semantics as characterized previously. As before we examine the termination semantics of $P_{A}\|_{B}Q$ by means of a simple instance of our test process, $(P_{A}\|_{B}Q); R$.

Example 3.2 Consider the case when $\checkmark$ is a private asynchronous event, that is $\checkmark \in \alpha(P)$ or $\checkmark \in \alpha(Q)$ but not both, thus only one of P and Q is capable of successfully terminating. We use the same processes as in Example 3.1 for R and P, but now Q cannot terminate:

$$P \mathrel{\widehat{=}} a \rightarrow SKIP \quad \alpha(P) = \{a, \checkmark\}$$
$$Q \mathrel{\widehat{=}} b \rightarrow STOP \quad \alpha(Q) = \{b\}$$

The failure set for P was given in Example 3.1 and that for Q is as follows:

$$\mathcal{F}[\![b \rightarrow STOP]\!]e = \{(\langle\,\rangle, X) \mid b \notin X \wedge X \in \mathbb{P}(\Sigma)\} \cup \{(\langle b\rangle, X) \mid X \in \mathbb{P}(\Sigma)\}$$

The failure set for $P_{A}\|_{B}Q$ is as follows:

$$\begin{aligned}\mathcal{F}[\![(a \rightarrow SKIP)_{A}\|_{B}(b \rightarrow STOP)]\!]e = \quad & \{(\langle\,\rangle, X) \mid a \notin X \wedge b \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle a\rangle, X) \mid \checkmark \notin X \wedge b \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle a, \checkmark\rangle, X) \mid b \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle b\rangle, X) \mid a \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle b, a\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle a, b\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle a, \checkmark, b\rangle, X) \mid X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle a, b, \checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\} \\ & \cup \{(\langle b, a, \checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma)\}\end{aligned}$$

Again a $\checkmark$ occurs other than at the end of a trace, e.g., $\langle a, \checkmark, b\rangle$. This failure set is also the failure set for the following process:

$$(a \rightarrow (\checkmark \rightarrow b \rightarrow STOP \,\Box\, b \rightarrow SKIP)) \,\Box\, (b \rightarrow a \rightarrow SKIP)$$

Therefore, we have the following process equivalence:

$$(a \rightarrow SKIP)_{A}\|_{B}(b \rightarrow STOP) \;\equiv\; (a \rightarrow (\checkmark \rightarrow b \rightarrow STOP \,\Box\, b \rightarrow SKIP)) \,\Box\, (b \rightarrow a \rightarrow SKIP) \tag{10}$$

Here the semantics for $_{A}\|_{B}$ gives rise to an undesirable equivalences because of the occurrence of the $\checkmark$ outside of *SKIP*. Now substituting (10) into our test process we have the following equivalence:

$$\begin{aligned}((a \rightarrow SKIP)_{A}\|_{B}(b \rightarrow STOP)); R \equiv (a \rightarrow ((b \rightarrow R) \,\Box\, R) \,\Box\, b \rightarrow a \rightarrow R) \\ \sqcap ((a \rightarrow R) \,\Box\, (b \rightarrow a \rightarrow R))\end{aligned} \tag{11}$$

Proof So substituting $(a \rightarrow SKIP)_{A}\|_{B}(b \rightarrow STOP)$ into the test process $P /\!\!/ Q; R$ we have:

$$\begin{aligned}&((a \rightarrow SKIP)_{A}\|_{B}(b \rightarrow STOP)); R \\ &\quad\equiv ((a \rightarrow (\checkmark \rightarrow b \rightarrow STOP \,\Box\, b \rightarrow SKIP)) \,\Box\, (b \rightarrow a \rightarrow SKIP)); R && \text{[by (10)]} \\ &\quad\equiv ((a \rightarrow ((\checkmark \rightarrow b \rightarrow STOP) \,\Box\, b \rightarrow SKIP)); R) \,\Box\, ((b \rightarrow a \rightarrow SKIP); R) && \text{[; /}\Box\text{ Distribution]} \\ &\quad\equiv (a \rightarrow ((\checkmark \rightarrow b \rightarrow STOP) \,\Box\, b \rightarrow SKIP); R) \,\Box\, (b \rightarrow a \rightarrow R) && \text{[; /}\rightarrow\text{ Distribution]}\end{aligned}$$

Then given that $(\checkmark \rightarrow P \,\Box\, Q); R \equiv R \sqcap (R \,\Box\, (Q; R))$ which is a variant of the general law $(P \,\Box\, SKIP); Q \equiv Q \sqcap ((P; Q) \,\Box\, Q)$, we have:

$$\begin{aligned}
&\equiv (a \rightarrow (R \sqcap (R \,\Box\, b \rightarrow R))) \,\Box\, (b \rightarrow a \rightarrow R) && [(P \,\Box\, SKIP); Q \equiv Q \sqcap ((P; Q) \,\Box\, Q)] \\
&\equiv ((a \rightarrow R) \sqcap (a \rightarrow (b \rightarrow R \,\Box\, R))) \,\Box\, (b \rightarrow a \rightarrow R) && [\sqcap/\rightarrow \text{ Distribution}] \\
&\equiv (a \rightarrow ((b \rightarrow R) \,\Box\, R) \,\Box\, b \rightarrow a \rightarrow R) \sqcap ((a \rightarrow R) \,\Box\, (b \rightarrow a \rightarrow R)) && [\sqcap/\Box \text{ Distribution}]
\end{aligned}$$

This completes the proof. □

Clearly, this equivalence is not what would be expected from the *lhs* process, and is again undesirable due to the introduction of pure non-determinism. In this example $_A\|_B$ has race termination semantics, and the single ✓ occurred in the middle of a trace.

For this operator it is possible that even with race termination semantics the ✓ will only occur at the end of the traces, this is not the case with |||. This situation arises with $_A\|_B$ but not with ||| because in the former there is the possibility of synchronous events covering up the problems, but with the latter there are never any synchronous events which can do this, we illustrated this with the following example.

Example 3.3 Consider the following process, where $A = \{a, b, \checkmark\}$ and $B = \{c, b\}$. Now using the standard set of CSP laws for $_A\|_B$ we can expand the process and eliminate $_A\|_B$ as follows:

$$\begin{aligned}
&(a \rightarrow b \rightarrow SKIP)\,{}_A\|_B (c \rightarrow b \rightarrow STOP) \\
&\quad \equiv (a \rightarrow ((b \rightarrow SKIP)\,{}_A\|_B (c \rightarrow b \rightarrow STOP))) \,\Box\, (c \rightarrow ((a \rightarrow b \rightarrow SKIP)\,{}_A\|_B (b \rightarrow STOP))) \\
&\quad \equiv (a \rightarrow c \rightarrow ((b \rightarrow SKIP)\,{}_A\|_B (b \rightarrow STOP))) \,\Box\, (c \rightarrow a \rightarrow ((b \rightarrow SKIP)\,{}_A\|_B (b \rightarrow STOP))) \\
&\quad \equiv (a \rightarrow c \rightarrow b \rightarrow (SKIP\,{}_A\|_B STOP)) \,\Box\, (c \rightarrow a \rightarrow b \rightarrow (SKIP\,{}_A\|_B STOP)) \\
&\quad \equiv (a \rightarrow c \rightarrow b \rightarrow SKIP) \,\Box\, (c \rightarrow a \rightarrow b \rightarrow SKIP) \qquad (12)
\end{aligned}$$

The synchronous event b forces the two processes to synchronize just before the left hand side (*lhs*) one terminates, and since the *rhs* one does nothing else after the b the ✓ only appears at the ends of the traces and the undesirable effects of race termination semantics are covered up. Note that if the *rhs* process could perform another event after the b then the associated problems of this type of semantics would occur since the ✓ would appear before this additional event in one of the traces.

Thus we have shown that the mixed parallel operator has race termination semantics under these circumstances, and it is easy to demonstrate that it has synchronous termination semantics when both subprocesses can terminate, see [How05]. This is not its natural termination semantics, which we claimed should be either synchronous or asynchronous. It is also the case that ✓s do not only occur at the ends of its traces but in the middle as well. Therefore, we need to replace this operator with one that has the natural semantics of ✓s only occurring at the end of a trace.

4. A CSP model with improved termination semantics

We begin to develop our solution to this termination problem by the definition of a failure-divergence semantic model for CSP that includes an improved treatment of termination. This is to be achieved by adding a successful termination axiom to the existing model.

We first formalize what types of processes and traces the ✓-requirement should be applied to by defining four termination axioms. Each one reflecting a different notion of what it means for a process to be *well-behaved* with respect to termination. The four semantic models that are produced by adding each of the four termination axioms to the existing process axioms are then analysed. The best fit is then used as our new model for CSP, which we shall call N_T. Next we prove that the existing language of CSP, except the asynchronous and mixed parallel operators, is well-defined and well-behaved in this new model N_T.

The definition of replacements for the asynchronous and mixed parallel operators with correct termination semantics, i.e., satisfying the termination axiom, is the subject of Sect. 5.

4.1. Successful termination axioms

We now define a new termination axiom that will be added to the existing CSP process axioms (see Appendix A) (D1)–(N5). The intention in doing this is to exclude from the failure-divergence model all processes which do not satisfy the ✓-requirement. In effect a sub-model of the existing failure-divergence model is characterized, and within this new sub-model all process denotations satisfy the ✓-requirement.

In attempting to define an axiom which captures the semantics of ✓ as representing successful termination the following issues need to be resolved:

- Should the ✓-requirement apply only to those processes which can never diverge, i.e., $\mathcal{D}[\![P]\!]e = \varnothing$, 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 ✓-requirement in relation to divergence because in the original CSP model N (see [BR85] and Appendix A), a diverging process can perform any and every trace; in particular it can perform traces which do not satisfy the ✓-requirement. Hence it is necessary to decide to which processes and which traces the ✓-requirement should be applied.

The potential responses to just these two questions indicate that there are at least three main types of termination axioms. The three cases we consider are the following.

1. Applies only to non-divergent processes.
2. Applies to divergent and non-divergent processes.
3. Applies only to the non-divergent traces of both divergent and non-divergent processes.

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

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

If the divergent set of a process is empty and $s^\frown\langle\checkmark\rangle$ 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 ✓ has occurred.

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

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

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

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

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

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

4.2. Comparison of the termination axioms

The (TA1) termination axiom 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. The (TA2) and (TA3) termination axioms reflect 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 when it is not actually diverging, but when it is diverging it is allowed to *misbehave* with respect to successful termination. The (TA4) axiom reflects the view that even if a process can diverge then it is still required to be well-behaved with respect to successful termination even if it is actually diverging.

(TA4) is incompatible with the existing CSP axioms since it excludes divergent processes and we choose to abandoned it for that reason. (TA1), (TA2) and (TA3) are compatible with the existing CSP process axioms and so we are left with the choice of deciding between them.

One 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. Which is intuitively appealing because given the disastrous effect of divergence, the fact that a process also violates the ✓-requirement, could be argued is irrelevant in comparison. The overriding reason for not adopting it is that it is too weak to enforce the ✓-requirement without modifying the semantics of ‖, as is illustrated by the following example.

Example 4.1 Consider the following processes:

$$P \mathrel{\widehat{=}} (a \to SKIP) ||| (b \to c \to \bot)$$
$$Q \mathrel{\widehat{=}} (a \to SKIP) ||| (b \to d \to \bot)$$

Then we have the following equivalences:

$$P \equiv \quad a \to (\checkmark \to b \to c \to \bot \,\Box\, b \to (\checkmark \to c \to \bot \,\Box\, c \to \bot)) \\ \Box\, b \to (a \to (\checkmark \to c \to \bot \,\Box\, c \to \bot) \,\Box\, c \to \bot) \tag{13}$$
$$Q \equiv \quad a \to (\checkmark \to b \to d \to \bot \,\Box\, b \to (\checkmark \to d \to \bot \,\Box\, d \to \bot)) \\ \Box\, b \to (a \to (\checkmark \to d \to \bot \,\Box\, d \to \bot) \,\Box\, d \to \bot) \tag{14}$$

Now both of these processes trivially satisfy (TA1) since they both diverge, but the following process does not:

$$P \| Q \;\equiv\; a \to (\checkmark \to b \to STOP \,\Box\, b \to SKIP) \,\Box\, b \to a \to SKIP \tag{15}$$

This is because the following is true:

$$\langle b \rangle \neq \langle\,\rangle \;\wedge\; D = \varnothing \;\wedge\; (\langle a, \checkmark \rangle, \varnothing) \in F \;\wedge\; (\langle a, \checkmark, b \rangle, \varnothing) \in F$$

This means that if (TA1) is added to the existing process axioms it would not be strong enough by itself to ensure that processes of the form $P \| Q$ satisfy our weak version of the ✓-requirement even in the cases when both P and Q do.

We therefore reject the (TA1) termination axiom. However, the above problem does not arise with (TA2) or (TA3) because the P and Q processes used in Example 4.1 do not satisfy (TA2) or (TA3), since for both P and Q we have the following:

(TA2) $\quad \langle b \rangle \neq \langle\,\rangle \;\wedge\; (\langle a, \checkmark, b \rangle, \varnothing) \in F \;\wedge\; \langle a, \checkmark \rangle \notin D$

(TA3) $\quad \langle b \rangle \neq \langle\,\rangle \;\wedge\; (\langle a, \checkmark, b \rangle, \varnothing) \in F \;\wedge\; \langle a \rangle \notin D$

So both versions eliminate the processes P and Q, thus they cannot be used as a counter example.

We are therefore left with the choice between (TA2) and (TA3). The suitability of (TA2) and (TA3) as candidates is reflected in the fact that:

- They enforce the ✓-requirement on all non-divergent traces of a process irrespective of whether it diverges or not, which (TA1) does not. Which from a subjective point of 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 $s^\frown\langle\checkmark\rangle^\frown t$ ($t \neq \langle\,\rangle$) then we now know that it is diverging, whereas with out 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 ✓, 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 [How05]) 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:

$$\checkmark \to \bot$$

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

To summarise, the main reasons for choosing (TA3) are that it solves the termination problem, most closely captures our intuitive views of termination and achieves this with only a minimal modification to the original semantics. In particular, (TA3) 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, i.e. satisfy 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 TA3 also maintains the original view that a diverging process cannot recover by performing a ✓. Henceforth, (TA3) will be referred to simply as the *Termination* axiom (T1):

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

4.3. The new model for CSP

The definition of the new model for CSP N_T is achieved by adding the Termination axiom (T1) to the existing CSP process axioms (D1)–(N5), see Appendix A. Hence in N_T the failure and divergence sets of a process are defined as for N, except that they must satisfy the process axiom (T1) as well as (D1)–(N5). The semantic functions $\mathcal{F}$ and $\mathcal{D}$ for N_T are the same as for N.

We define the space of processes $(PROC_T, \sqsubseteq)$ for N_T, where $PROC_T$ is the sub-set of processes of $PROC$ that satisfy (T1) in addition to (D1)–(N5). The order relation $\sqsubseteq$ is the same as for N. Therefore, the space of processes together with the ordering, i.e., $(PROC_T, \sqsubseteq)$ is a partially ordered set, with least element $\bot$. As in N, $\bot$ denotes the divergent process and the maximal elements are again the deterministic processes.

As with $(PROC, \sqsubseteq)$, it is necessary to prove that the space of processes $PROC_T$ under $\sqsubseteq$ is closed under the union of arbitrary non-empty sets of processes and the intersection of *directed* sets of processes, i.e., they are both processes. This means that $(PROC_T, \sqsubseteq)$ is a complete semi-lattice.

The proof of this requires showing that the union of arbitrary non-empty sets of processes and the intersection of directed sets of processes satisfy (D1)–(T1). Since $PROC_T \subseteq PROC$ it is only necessary to do this for (T1), the proofs for both cases are straightforward and similar to those given in [Bro83, BHR85], see [How05].

4.4. The effect of the Termination axiom on CSP

The result of adding the Termination axiom (T1) to the existing process axioms is that certain failure-divergence sets which denoted processes in N do not do so in N_T. For this reason we must return to the language of CSP which unless is restricted and modified in some way will continue to produce processes which are denoted by failure-divergence sets which do not satisfy (T1), i.e., are not well-defined processes in N_T.

In the previous section, we saw that the two parallel operators $|||$ and $_A\|_B$ are capable of producing processes which do not satisfy (T1). It is now necessary to show that the basic processes satisfy (T1) and also that the other operators in the language satisfy (T1) provided their subprocesses do. Further it is also necessary to show that the CSP operators are well-behaved in N_T, i.e., are strict, monotonic, continuous and distributive. First we define the "(T1) safe" language of CSP for N_T that we shall now refer to as CSP_T, and is defined as follows:

$$P ::= \bot \mid STOP \mid SKIP \mid a \to P \mid P \sqcap P \mid P \,\Box\, P \mid$$
$$P \| P \mid P; P \mid P \backslash b \mid f(P) \mid f^{-1}(P) \mid \mu p.F(p) \mid p$$

where $a \in \Sigma - \{\checkmark\}$, $b \in \Sigma$ and f is an alphabet transformation which satisfies (ATC):

(ATC) $$f \in \{f' \mid f'(x) = \checkmark \Leftrightarrow x = \checkmark\}$$

In the definition of the CSP term $F(p)$, only the above processes and operators are permitted.

We can now state the theorem for our new model N_T and its language CSP_T as follows:

Theorem 4.1 *The processes and operators of* CSP_T *are* well-defined *and* well-behaved *in* N_T.

Proof For the basic processes $\bot$, *STOP* and *SKIP* proofs are immediate from their definition. The proofs of $P \sqcap Q$ and $P \,\Box\, Q$ are straightforward. The proofs of $P \| Q$, $P; Q$, $P \backslash a$ and $\mu p.F(p)$ do not require additional constraints and are given in [How05]. The processes $a \to P$, $f(P)$ and $f^{-1}(P)$ do not satisfy (T1) unless the additional constraints given above are introduced, these proofs are also given in [How05]. □

5. Defining replacement parallel operators for CSP$_T$

We now introduce three types of parallel operators each with a different type of parallel termination semantics, namely race, synchronous and asynchronous. Two of these new parallel operators with race and synchronous termination semantics are defined using a generalized parallel operator. Note that from the analysis of the three existing parallel operators performed in Sect. 3.2, that none of them has asynchronous termination semantics. Therefore, if we wish to have a parallel operator with asynchronous termination semantics it will be necessary to define a new type of parallel operator.

5.1. The generalized parallel operator

The generalized parallel operator has the set of events that the combined processes are required to synchronize on, i.e. the synchronization set, given as an explicit parameter of the operator. This is in contrast to the existing operators where the synchronous events are fixed or implied either relative to the alphabets of the processes being composed as in $_A\|_B$ or irrespective of their alphabets as in $\|$ and $|||$. We denote our generalized operator using the following notation:

$$P \|_\Omega Q \quad \text{where } \varnothing \subseteq \Omega \subseteq \Sigma$$

The intended semantics of $P\|_\Omega Q$ is that P and Q are composed in parallel and are required to synchronize on every event in Ω, irrespective of their alphabets. Events which are not in Ω but which can be performed by either P or Q or both are asynchronous events. Similar forms of generalized parallel operator have previously been introduced by various authors into CSP [FDR03, Ros98, TW97] and Timed CSP [Dav93]. The semantic functions for our version $P\|_\Omega Q$ are given in Appendix A.

5.2. Race termination semantics

We now define a parallel operator with race termination semantics that also satisfies the ✓-requirement, i.e., (T1). The race termination semantics parallel operator is denoted as follows:

$$P \lbag\!\rbag_\Theta Q \quad \text{where } \varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$$

This operator is similar to $\|_\Omega$ in the sense that the synchronization set Θ is explicitly stated, but ✓ is never a member. The intuitive meaning of $P \lbag\!\rbag_\Theta Q$ is that the two processes P and Q are composed in parallel and synchronize on all events in Θ and it has race termination semantics. Therefore, $P \lbag\!\rbag_\Theta Q$ terminates when P terminates asynchronously or when Q terminates asynchronously, and $P \lbag\!\rbag_\Theta Q$ fails to terminate if and only if P and Q both fail to terminate.

The definition of this new operator is based on the test process used in 3.2.2, that is $(P /\!\!/ Q); R$. The reason for using a version of the test process is that it is the simplest way to achieve the desired result. To see this consider the processes used in Example 3.1, when $|||$ was used for $/\!\!/$. It was shown there that in this context $P|||Q$ had race termination semantics and that the multiple ✓s which occurred when it was viewed in isolation no longer occurred. This was because the sequential composition with R in effect caused everything after a ✓ occurred to be aborted. So the new operator is defined using the test process with $\|_\Theta$ for $/\!\!/$ and $SKIP$ for R.

Definition 5.1 The race termination semantics parallel operator is defined as follows:

$$P \lbag\!\rbag_\Theta Q \;\widehat{=}\; (P\|_\Theta Q); SKIP \quad \text{where } \varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$$

The restriction that ✓ is not allowed to be a member of Θ ensures that ✓ is performed asynchronously and that any ✓ performed by either P or Q is visible to $SKIP$, which after it observes a ✓ proceeds to execute, i.e., just terminates. So $P\|_\Theta Q$ could violate the ✓-requirement, but because it has been sequentially composed with $SKIP$, race termination semantics satisfying the ✓-requirement are produced.

As an example of the use of the race termination operator consider the process $P \lbag\!\rbag_\Theta Q$, when $\Theta = \varnothing$, in this case P and Q are intended to execute independently. However, a strict interpretation of race termination would require that when the first process terminates the other process is aborted, i.e., terminated. This would appear to imply that their executions are not completely independent. This apparent contradiction can be resolved if instead of interpreting race termination as meaning that the non-terminated process is aborted/terminated, we take a more pragmatic view and say that it is to be "henceforth ignored" and consequently that it will have no further interaction with the system. Obviously, in practice the simplest way to achieve this would be to terminate the process, but if the executions were really required to be independent, for example as in a distributed system, then a more elaborate approach would be required to achieve this.

5.3. Asynchronous termination semantics

To achieve asynchronous termination semantics for any form of parallel composition we must define a new operator which has this type of semantics since as was noted in 3.1, none of the existing parallel operators has asynchronous termination semantics. The new parallel operator is denoted as follows:

$$P /\!\!/_{\Theta} Q \quad \text{where } \varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$$

This operator is similar to $\|_{\Omega}$ in the sense that the synchronization set Θ is explicitly stated, but $\checkmark$ is never a member. The intuitive meaning of $P /\!\!/_{\Theta} Q$ is that the two processes P and Q are composed in parallel and synchronize on all events in Θ and has asynchronous termination semantics. Therefore, $P /\!\!/_{\Theta} Q$ terminates when both P and Q have terminated asynchronously, and it fails to terminate if either P fails to terminate or if Q fails to terminate or both. The semantic functions for $P /\!\!/_{\Theta} Q$ are given in Appendix A.

5.4. Synchronous termination semantics

A generalized parallel operator with synchronous termination semantics can easily be defined using the existing generalized parallel operator $P\|_{\Omega} Q$ with the additional requirement that $\checkmark$ is always included in the synchronization set. The new operator is denoted as follows:

$$P\|_{\Delta} Q \quad \text{where } \{\checkmark\} \subseteq \Delta \subseteq \Sigma$$

The intuitive meaning of $P\|_{\Delta} Q$ is that the two processes P and Q are composed in parallel and synchronize on all events in Δ which always includes $\checkmark$, thus ensuring synchronous termination semantics. To distinguish this special version of the generalized operator which has synchronous termination semantics we shall denote the synchronization set by Δ, and define it by $\Delta = \Theta \cup \{\checkmark\}$ where $\varnothing \subseteq \Theta \subseteq \Sigma - \{\checkmark\}$.

5.5. The operational semantics of the new parallel operators

We now present the operational semantics for the three new parallel operators using a *labeled transition system*. The algebraic laws and comparison of the three operators can be found in [How05]. The style and notation of operational semantics we use is that of Roscoe's [Ros98], in particular we make use of:

- The special process term Ω to represent any process that already has terminated. Ω can perform no transitions.
- The special action τ that represents hidden events, for example, the resolution of a nondeterministic choice or the act of unwinding a recursion definition. We shall use it to represent the hidden successful termination of a process.

The *firing rules* for the three parallel operators are the same for non-$\checkmark$ events. So the following rules for $\backslash\!\backslash_{\Theta}$ are the same for $/\!\!/_{\Theta}$ and $\|_{\Delta}$:

$$\frac{P \xrightarrow{a} P', \quad Q \xrightarrow{a} Q'}{P \backslash\!\backslash_{\Theta} Q \xrightarrow{a} P' \backslash\!\backslash_{\Theta} Q'} \quad [a \in \Theta]$$

$$\frac{P \xrightarrow{a} P'}{P \backslash\!\backslash_{\Theta} Q \xrightarrow{a} P' \backslash\!\backslash_{\Theta} Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \backslash\!\backslash_{\Theta} Q \xrightarrow{a} P \backslash\!\backslash_{\Theta} Q'} \quad [a \notin \Theta]$$

$$\frac{P \xrightarrow{\tau} P'}{P \backslash\!\backslash_{\Theta} Q \xrightarrow{\tau} P' \backslash\!\backslash_{\Theta} Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \backslash\!\backslash_{\Theta} Q \xrightarrow{\tau} P \backslash\!\backslash_{\Theta} Q'}$$

Table 1. Replacements for $\|$, $|||$ and $_A\|_B$ in N_T

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

The termination firing rules, that is, the firing rules for $\checkmark$ for the three operators are different and illustrate the different ways in which the three parallel operators $\setminus\!\setminus_\Theta$, $/\!/_\Theta$ and $\|_\Delta$ successfully terminate. The firing rules are as follows:

For race termination semantics:

$$\frac{P \xrightarrow{\checkmark} P'}{P \setminus\!\setminus_\Theta Q \xrightarrow{\checkmark} \Omega} \qquad \frac{Q \xrightarrow{\checkmark} Q'}{P \setminus\!\setminus_\Theta Q \xrightarrow{\checkmark} \Omega}$$

The rules represent the fact that $P \setminus\!\setminus_\Theta Q$ terminates if either P or Q terminates.

For asynchronous termination semantics:

$$\frac{P \xrightarrow{\checkmark} P'}{P /\!/_\Theta Q \xrightarrow{\tau} \Omega /\!/_\Theta Q} \qquad \frac{Q \xrightarrow{\checkmark} Q'}{P /\!/_\Theta Q \xrightarrow{\tau} P /\!/_\Theta \Omega}$$

Here the successful termination of one of the processes is a hidden event and is represent by τ. It is also necessary to define a firing rule for the case when both processes have been transformed into Ω as follows:

$$\frac{}{\Omega /\!/_\Theta \Omega \xrightarrow{\checkmark} \Omega}$$

The rules represent the fact that $P /\!/_\Theta Q$ terminates only after both P and Q have terminated asynchronously.

For synchronous termination semantics:

$$\frac{P \xrightarrow{\checkmark} P' \quad Q \xrightarrow{\checkmark} Q'}{P \|_\Delta Q \xrightarrow{\checkmark} \Omega}$$

The rule represents the fact that $P \|_\Delta Q$ terminates only when P and Q terminate synchronously.

5.6. The language of CSP$_T$

Before we can add the three new parallel operators to CSP$_T$ we must ensure that they are well-defined and well-behaved in N_T, which we now do in the following theorem.

Theorem 5.1 The three parallel operators $\setminus\!\setminus_\Theta$, $/\!/_\Theta$ and $\|_\Delta$ are *well-defined* and *well-behaved* in N_T.

Proof For details see [How05]. □

Since, we have proved that $\setminus\!\setminus_\Theta$, $/\!/_\Theta$ and $\|_\Delta$ satisfy (T1), they can be added to CSP$_T$, the language for the new model N_T that was defined in Sect. 4.4. The final version of CSP$_T$ is then defined as follows:

$$\begin{aligned} P ::= {} & \bot \mid STOP \mid SKIP \mid a \rightarrow P \mid P \sqcap P \mid P \,\Box\, P \mid \\ & P; P \mid P \backslash b \mid f(P) \mid f^{-1}(P) \mid \mu p.F(p) \mid p \mid \\ & P \setminus\!\setminus_\Theta P \mid P /\!/_\Theta P \mid P \|_\Delta P \end{aligned}$$

where $a \in \Sigma - \{\checkmark\}$, $b \in \Sigma$ and in the definition of $F(p)$, only the above processes and operators can be used. In addition, f must satisfy the following alphabet transformation constraint (ATC):

$$\text{(ATC)} \qquad f \in \{f' \mid f'(x) = \checkmark \Leftrightarrow x = \checkmark\}$$

The three new operators can be used as replacements for $\|$, $|||$ and $_A\|_B$, the details are given in Table 1. So finally, by adding these three new operators with different termination semantics to CSP$_T$ we have achieved our aim of providing flexible parallel termination semantics for CSP.

5.7. Distinguishing between synchronous and asynchronous termination

In our model it is possible to distinguish between the two parallel operators with synchronous ($\|_\Delta$) and asynchronous ($/\!/_\Theta$) termination semantics when one of the processes or sub-process being composed in parallel is of the form $P \,\Box\, SKIP$. This difference can be seen in the following two *expansion* (or *step*) laws for the two operators, taken from [How05].

$$\text{(L6.27)} \qquad SKIP /\!/_\Theta ((x : X \to Px) \,\Box\, SKIP) \equiv SKIP \sqcap (SKIP \,\Box\, (x : X \cap \overline{\Theta} \to (SKIP /\!/_\Theta Px)))$$

$$\text{(L6.44)} \qquad SKIP \|_\Delta ((x : X \to Px) \,\Box\, SKIP) \equiv SKIP \,\Box\, (x : X \cap \overline{\Delta} \to (SKIP \|_\Delta Px))$$

In our model we are able to make this distinction between the two forms of parallel operator since we maintain the original view of ✓, and as a result we allow processes of the form $P \,\Box\, SKIP$. This form of process is ruled out in Roscoe's model by the (F4) process axiom and in Hoare and He's UTP by the healthiness condition (CSP4). However, in [How05] we prove that synchronous termination and asynchronous termination are equivalent if we add the following (US) axiom, which is equivalent to Roscoe's (F4) axiom, to our model.

$$\text{(US)} \qquad (s^\frown\langle\checkmark\rangle, \varnothing) \in F \wedge \checkmark \notin X \Rightarrow (s, X) \in F$$

This has the same effect as Roscoe's (F4) axiom of eliminating unguarded *SKIP*s from the choice operator. This result is formalized in the following theorem regarding the equivalence of the two parallel operators.

Theorem 5.2

$$\mathrm{US}(P) \wedge \mathrm{US}(Q) \quad \Rightarrow \quad P /\!/_\Theta Q \;\equiv\; P \|_\Delta Q$$

Proof See [How05] for details. □

As a result of this we believe that any CSP model that has (CSP4) as a law or a process axiom similar to (F4) would result in asynchronous termination being equivalently to synchronous termination in that model. A more complete comparison of all three parallel operators can be found in [How05], along with a selection of algebraic laws for each one.

6. Comparison of solutions

In this section, we first compare our solution with those of Tej and Wolff's [TW97] and Roscoe's [Ros98] since these are achieved by modifications to the failure-divergence model. Secondly, we discuss the relationship between our solution and Hoare and He's [HJ98] UTP presentation of CSP. Next we discuss the advantages of our solution in relation to those of Roscoe's and Hoare and He's. Finally, we discuss how parallel termination is dealt with in several other process algebras.

6.1. Comparison of failure-divergence model based solutions

In choosing our termination axiom (T1) we have taken the view that a process that is capable of diverging is unlikely to be well-behaved with respect to termination. This contrasts with the view taken by Tej and Wolff, and Roscoe, who do require a divergent process to stop diverging and successfully terminate if it does a ✓. This is reflected in the fact that both their models have applied the ✓-requirement to both non-divergent and divergent traces of a process. Note that in the original semantics and ours a diverging process cannot be stopped from diverging by performing a ✓, thus to some extent they have a "less chaotic" view of divergence.

In essence they have chosen to adopt the approach represented by our (rejected) (TA4) termination axiom of Sect. 4.1. Thus the various modifications they have made illustrate what changes would be required if this axiom had been chosen. In comparing the individual axioms chosen by the various approaches it is interesting to note the following.

- Tej and Wolff [TW97] discovered that the sequential composition operator $P;Q$ did not satisfy (N2), and therefore, was not a well-defined process in N. The proof they used to show that $P;Q$ did not satisfy (N2) was based on a number of assumptions about the processes P, Q and $P;Q$. However, these assumptions are inconsistent with our termination axiom (T1), thus our new model N_T solves this problem with $P;Q$, by excluding the types of processes which could cause it, see [How05] for details.
- Tej and Wolff's (v*) axiom is the same as Roscoe's (D1), which reflects their shared requirement that ✓s are only allowed to be final events in divergent traces as well as in non-divergent traces. This is not the case in our approach, hence we do not need a new axiom to capture this, but can use the original (D1).
- The requirement for non-divergent traces to be "front-tick-free" captured by Tej and Wolff's (*) axiom, is captured by Roscoe in restricting certain traces to the following set:

$$\Sigma^{*\checkmark} = \Sigma^* \cup \{ s^\frown\langle\checkmark\rangle \mid s \in \Sigma^* \}$$

- Our termination axiom (T1) can be read as stating that the only way a process can do the trace $s^\frown\langle\checkmark\rangle^\frown t$ is if it had started diverging before the ✓. Roscoe's (D3) axiom similarly means that the only way a process can have the divergent trace $s^\frown\langle\checkmark\rangle$ is if it had started diverging before the ✓. Thus there is a similarity in the way that these two axioms permit a process to behave in (what could be considered) a strange way, provided the process is already diverging.

With respect to the form of parallel termination semantics adopted both Tej and Wolff [TW97], and Roscoe [Ros98] define the three original parallel operators in terms of a generalized parallel operator $P[|X|]Q$ (or $P\|_X Q$), where P and Q synchronize on events in X. Tej and Wolff's version uses synchronous termination semantics ($\checkmark \in X$) to define the three operators, whereas Roscoe uses distributed termination (asynchronous termination semantics) ($\checkmark \notin X$) to define the three operators. In this paper, we have also defined the three original parallel operators using a generalized operator. However, instead of being restricted to one operator with only one form of termination semantics, we can use either of the three forms of parallel operator with race, asynchronous or synchronous terminations semantics.

One of the main differences between our solution and Roscoe's [Ros98] is the different views we take of successful termination, this is reflected in how ✓ is dealt with. Roscoe views it as a "signal" that the environment can observe but not refuse, i.e., the environment does not control termination, and we maintain its original meaning that the environment can do both, and hence can control termination. We believe that the original view of ✓ still has merit since there are occasions when the environment does control termination, for example, when an "off switch" is used to terminate some device. In this case the environment simultaneously controls and observes termination, by means of the off switch. In the original model and ours ✓ is the event used for this purpose, but if ✓ is to be viewed solely as signalling termination then it seems that some other event needs to be used to represent the "off switch" event. This event could either be some normal event ($\in \Sigma$) or a specially designated "off switch/terminate" event $\circlearrowleft$. However, the introduction of yet another special event would clearly complicate[3] the situation and we shall therefore leave this for future investigation.

6.2. Relationship between our solution and UTP

In Hoare and He's *unifying theory of programming* (UTP) [HJ98], they define *healthiness conditions* that characterize the sets of relations that represent different programming paradigms. Using this approach they define CSP processes as those relations that satisfy the *reactive* processes healthiness conditions and the five specific CSP ones.

[3] Clearly, the idea of introducing yet another special event, given the problems caused by ✓, would be a very daunting one. However, if this "terminate" event $\circlearrowleft$ were introduced then it could be used in some circumstances as an alternative to $P \,\Box\, SKIP$, given the objections to it that a process should not offer a normal event at the same time as terminating. By defining the process $TERMINATE \mathrel{\widehat{=}} \circlearrowleft \rightarrow SKIP$, where $\circlearrowleft$ would only be used in this process and as with ✓ it could not be hidden, then $P \,\Box\, TERMINATE$ could be used as an alternative to $P \,\Box\, SKIP$. Other issues would also have to be resolved for example, would $TERMINATE; P \equiv TERMINATE$ hold.

A significant difference between the failure-divergence based models for CSP and the UTP approach is that UTP does not use ✓ to represent successful termination, but the observation variables ok' and $wait'$, i.e., if $ok' \wedge \neg wait'$ holds then the process has successfully terminated.

In relation to the individual CSP healthiness conditions the most relevant to our work are clearly (CSP4) and (CSP5):

$$\begin{array}{ll} \text{(CSP4)} & P = P; SKIP \\ \text{(CSP5)} & P = P \,|||\, SKIP \end{array}$$

(CSP4) means the value of ref' is irrelevant after a CSP process terminates and (CSP5) means that if a process is deadlocked and refusing some set of events offered by its environment, then it would still be deadlocked if the environment offered it fewer of these events.

(CSP4) is equivalence (3) from Sect. 1.1 and as we have discussed this fails to hold if $P = Q \,\Box\, SKIP$. Thus, since both UTP and Roscoe require this to be a law, it was necessary for their models to be constrained accordingly. Since we do not require (CSP4) to hold in our model N_T, or have adopted Roscoe's view of ✓ as a signal, it is not necessary to place similar constraints, such as adding the process axiom (US) that is equivalent to (F4), to our model. Thus, CSP_T includes processes that do not satisfy this healthiness condition. In [How05] we show that if we added the (US) axiom to CSP_T and in effect exclude processes of the form $P \,\Box\, SKIP$ then the two parallel operators with asynchronous ($/\!/_\Theta$) and synchronous ($\|_\Delta$) termination semantics are equivalent. With regard to (CSP5), of our three generalized parallel operators with race, asynchronous and synchronous termination semantics, only the synchronous termination parallel operator $\|_\Delta$, satisfies (CSP5), see [How05] for details.

The *desired* equivalences that were identified for the two existing parallel operators $|||$ and ${}_A\|_B$ in Sect. 3.2, can be viewed as parallel termination healthiness conditions. For example, the desired equivalence (7) from Sect. 3.2.2 for $|||$:

$$(a \rightarrow SKIP) \,|||\, SKIP \;\equiv\; a \rightarrow SKIP$$

is an instance of (CSP5), however, Hoare and He [HJ98] use it to capture a different condition. In conclusion, it was our desire to satisfy these parallel termination healthiness conditions that ultimately led to the development of our new language CSP_T and its model N_T.

6.3. Advantages of our solution

We believe that our solution to the termination problem has two advantages over the other two main solutions of Roscoe [Ros98] and Hoare and He [HJ98]. Firstly, it is the simplest, in the sense that it requires fewer modifications to the original semantics. Secondly, it provides greater flexibility in the choice of termination semantics for the parallel operators provided.

With regard to the complexity of the various solutions, ours is the most similar to the original semantic model, it is achieved with the least modifications and thus is the simplest. In particular, we have retained the existing process axioms, added a single "termination axiom", imposed minor syntactic restrictions on the non-parallel operators and defined three new parallel operators with different termination semantics as possible replacement for the existing parallel operators. In comparison Roscoe's solution involves adding two new axioms (F4), (D3) and modifying (D1). In addition, ✓ is no longer a normal event, but a "signal" that must be "protected" when defining refusals. This altered view of ✓ also means that now a diverging process stops diverging and successfully terminates if it performs a ✓. Roscoe also provides various parallel operators all with "distributed" (asynchronous) termination semantics. Hoare and He's UTP approach uses an alphabetized relational calculus in combination with healthiness conditions to provide a unifying semantic framework for different programming paradigms. This is a very different approach to the failure-divergence model based solutions of ours and Roscoe's, although the failure-divergences semantics for CSP can be recovered using this approach. The UTP solution is part of a much grander project than simply solving the termination problems of CSP, and thus as a result is more complex than the failure-divergence model based solutions.

As part of our solution we have defined three generalized parallel operators each with a different from of terminations semantics – race, asynchronous and synchronous. Consequently, this provides the flexibility to select the most appropriate type of parallel termination semantics for the type of parallel process under consideration. For example, we can define ||| with asynchronous termination, $_A\|_B$ and $\|_X$ with synchronous termination and an *enslavement* (master/slave) operator with race termination, where its termination is solely determined by the termination of the master process. In contrast, Roscoe provides a collection of parallel operators that all use "distributed" (asynchronous) termination semantics; and it is unlikely that he would add this flexibility since he explicitly rejects race and synchronous termination semantics (see [Ros98, pp. 139–140]). In Hoare and He's UTP version of CSP it appears that the two healthiness conditions (CSP4) and (CSP5) restrict the forms of parallel termination semantics that can be distinguished and defined. In our model the consequence of requiring (CSP4) to hold is that it is not possible to distinguish between synchronous and asynchronous termination. This also appears to be the case for the UTP defined parallel operators as their definitions appear to be compatible with either synchronous or asynchronous termination semantics. In relation to (CSP5) of our three parallel operators only the one with synchronous termination satisfies it. Again it seems unlikely that Hoare and He would choose to provide the flexibility of being able to choose different forms of parallel termination semantics by introducing operators that did not satisfy their chosen set of healthiness conditions.

In conclusion, we believe that an advantage of the simplicity of our solution is that it is in some sense a "minimal solution" within the failure-divergence framework, in that any other solution to the problem would involve more constraints than ours. For example, if we were to add the (US) axiom (equivalent to Roscoe's (F4) and UTP's (CSP4)) then in our model we would no longer be able to distinguish between synchronous and asynchronous termination semantics, which we believe is the case for the models that include this constraint. Therefore, since our solution offers this freedom we believe that it could provide a useful basis to investigate more restricted versions of the model and language that would be required to satisfying additional or different healthiness conditions. And finally, neither Roscoe's nor Hoare and He's versions of CSP currently provide the flexibility of our solution with respect to parallel termination semantics.

6.4. Termination semantics in other process algebras

Although the termination semantics of ||| is undesirable in (untimed) CSP in Timed CSP it has been used to advantage by Davies [Dav93], Davies and Schneider [DS89] and Schneider [Sch90] to define event interrupt and timed interrupt (*tireout*) operators. The use of ||| in the definition of these operators is combined with ";" in a similar way to our test process, thus ensuring ✓s only occur at the end of traces. The problems of ||| and its use in Timed CSP suggests that this form of ||| should not be used in CSP or Timed CSP explicitly but only as a means to define other operators. A partial solution has been introduced into Timed CSP by Davies and Schneider [DS92] with the use of synchronized termination semantics in their generalized parallel operator $P[|X|]Q$. Their method is similar to ours, in particular they use $P[|X|]Q$ with $X = \{\checkmark\}$ to re-define |||. And in the definition of $_A\|_B$ they require ✓ to be implicitly included in A and B. Although they still use the old version of ||| to define the event and time interrupt operators and have not added a termination axiom. However, the authors are not aware of any modification of the models for Timed CSP developed by Reed [Ree88] and Reed and Roscoe [RR96], 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 ✓ in these models.

Aceto and Hennessy [AH92] have proposed a CCS [Mil89] and ACP [BW90] related process algebra, where they investigate successful termination as it relates to CCS and ACP. It is of general interest but is not directly applicable to the solution presented in this paper, due to the difference in approach of these two languages to CSP. For example CCS does not have a sequential composition operator ";" and uses the process *nil* to represent both successful termination and deadlock. ACP does have sequential composition and uses the symbol δ (corresponding to *nil* and *STOP*) to represent a deadlocked process and its absence to represent a successfully terminated one; ACP also has the "empty process" ϵ, which represents the immediately terminating process, similar to *SKIP*. ACP [BW90, pp. 75–76] also encounters problems with how to deal with the successful termination of its parallel operators $\|$ *merge* (interleaving) and $\underline{\|}$ *left merge*, it attempts to overcome this by the introduction of the termination operator $\surd$ which indicates whether or not a process has a termination option or not, returning ϵ or δ respectively.

7. Further work

An area for future investigation is how an improved treatment of successful termination can be incorporated into Timed CSP (TCSP) [Ree88]. Since there are several semantic models for TCSP (see [Ree88, Ree90, RR96]), Reed [Ree88] presents these models in a hierarchy. Hence, a complete solution would require the definition of a termination axiom for each model. Here we only consider the *Timed Failures-Stability* (TM_{FS}) model,[4] and present a candidate termination axiom for this model. Reed's original axioms for these models can be found in [Ree88]. If a termination axiom (TA) is added to a TCSP model then it would be necessary to prove that the new model, for example $(TM_{FS} + TA, d)$, 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 follow those of Reed's for TM_{FS} presented in [Ree88].

The Timed Failures-Stability model TM_{FS}, models processes by denoting them by sets of 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:

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

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

TA_{FS} 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 the $\checkmark$-requirement is satisfied, the $\checkmark$ occurred at the time of stability α and that from time α it can henceforth refuse all further events.

8. Conclusions

In this paper, we identified a problem with the treatment of parallel termination in the original semantics of CSP. Our solution to the problem entailed defining a modified failure-divergence model N_T for CSP, by the addition of a termination axiom to the original model. We then defined a new dialect CSP_T that was compatible with this new model. In particular, CSP_T included three new generalized parallel operators with *race*, *asynchronous* and *synchronous* termination semantics.

Several practical issues arise from the introduction of three generalized parallel operator each with a different form of termination semantics into CSP. In particular, are they applicable in practice due to implementation difficulties and determining which form of operator to use in a particular context.

We believe that for the vast majority of systems that asynchronous termination is probably the most appropriate and efficient, followed by synchronous termination which is likely to be appropriate for fewer types of systems. However, in the case of race termination semantics it is likely to be considered too inefficient in the vast majority of situations due to the overhead of dealing with the non-terminated processes.

An implementation of $P \setminus\!\setminus_{\Theta} Q$ would have to decide how to "ignore" the non-terminated process. For example, it could either terminate it immediately, delay its termination until convenient or allow it to run to completion and discard its results. If it is not terminated immediately then an implementation would be obliged to ensure that it was "ignored" and did not interact with the system. In practice, this is likely to require a rather complex and/or messy runtime procedure to deal with the necessary forced termination of the non-terminated processes. Examples of situations where this overhead associated with race termination might be acceptable or even desirable are when there is a need to find a solution as quickly as possible via competing processes or when the termination of the redundant losing processes allows their "freed up" resources to be re-allocated sooner and thus more efficient use made of them. This scenario might arise when performing repeated searches of large amounts of data, split between several processes. Here, as soon as the first search process completed the others would be terminated and the freed resources could be immediately re-used for the next search. This would seem more efficient than waiting for all processes to finish before starting the next search.

[4] Candidate termination axioms for several of the other models can be found in [How05].

The problem of how to resolve the choice about which one of the three forms of parallel termination to use would, we believe, largely depend on the context. For example, the decision may be based on which would be the most efficient or simplest to implement in a given context, e.g., asynchronous in a distributed system, synchronized in a shared memory system and race in web based searching. Alternatively, the decision might be based on requiring the operators to satisfy a particular collection of healthiness conditions such as those of UTP, in which case since synchronous and asynchronous termination are indistinguishable either would be satisfactory. It is hoped that the previous discussion regarding the types of systems the three operators are most appropriate for, in particular that with race termination semantics, would help to alleviate any uncertainty.

Finally, we believe the aims of the paper have been met, by providing a precise definition of the successful termination of a process, and solving the problem of the mismatch between the intuitive meaning of successful termination and how this notion was represented in the original failure-divergence semantic model. In addition, the aim of providing flexibility in the types of parallel termination semantics available has been achieved by the introduction of the three new parallel operators.

Acknowledgments

The authors gratefully acknowledge the advice and assistance of the following people Richard Bornat, Jonathan Bowen, Peter Burton, Michael Luck, Steve Reeves, Steve Schneider and Steve Winter. In addition, the authors would like to thank the anonymous referees for their suggestions and comments regarding the contents and structure of the paper.

Appendix A: Failure-divergence model for CSP

This appendix contains an introduction to CSP, details of notation, process axioms, the semantic functions for the *failure-divergence* model N for CSP. Plus, the semantic functions for the generalized parallel operator and the generalized parallel operator with asynchronous termination semantics for the model N_T for CSP_T.

A.1 Introduction to CSP

The following is a very brief introduction to the language of CSP, for a more detailed introduction see [Hoa85, Ros98]. The *standard* set of algebraic laws for CSP can be found in [BR85, Bro83, How05].

STOP is the *deadlocked* process that performs no events and does nothing. *SKIP* is the process that *successfully terminates* and then does nothing. $\perp$ is the *divergent* process that performs an unbounded number of (hidden) internal events. $a \rightarrow P$ is process that performs the event a and then behaves like P. $P \| Q$ is synchronous parallel composition and requires all events that are possible for P or Q be performed synchronously. $P \|\| Q$ is asynchronous (interleaving) parallel composition where all events that are possible for P or Q are performed asynchronously. $P_A \|_B Q$ is the mixed (alphabetized) parallel composition where all events that are common to P and Q are performed synchronously, other events are performed asynchronously. $P; Q$ is the sequential composition of P and Q. $P \Box Q$ is the *deterministic* choice between the two processes P and Q. $P \sqcap Q$ is the *non-deterministic* choice between the two processes P and Q. $P \backslash a$ is the process that behaves like P except that a events are hidden from the environment. The process $f(P)$ can perform the event $f(a)$ whenever P could have performed a, f is known as an *alphabet transformation*. The process $f^{-1}(P)$ can perform the event b whenever P could have performed the image event $f(b)$. Recursive processes are denoted by $\mu p.F(p)$ where p is a free variable of the process term $F(p)$.

A.2 Notation

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. 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, i.e., s is a prefix of t, meaning $s^\frown u = t$ for some u. $s < t$ is the proper *prefix* relation, meaning $s^\frown u = t$ for some $u \neq \langle\rangle$. $\#s$ is the length of the sequence s. a in s is sequence membership, i.e., a is in the sequence s. a i$\not{\text{n}}$ s is sequence non-membership, i.e., a is not in the sequence s. $\overline{X}$ 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 denote by P, Q, R and A, B, C denote their alphabets, i.e., sets of events $\mathbb{P}(\Sigma)$.

A.3 Process axioms

In this model a process P is denoted by an ordered pair $\langle F, D\rangle$, where F is the *failure set* of P, and D is the *divergence set* of P. A failure set F and divergence set D are any sets that satisfy the following conditions:

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

In addition, they must also satisfy the following *process axioms*:

$$s \in D \Rightarrow s^\frown t \in D \qquad (D1)$$
$$s \in D \Rightarrow (s^\frown t, X) \in F \qquad (D2)$$

$$(\langle\,\rangle, \varnothing) \in F \qquad (N1)$$
$$(s^\frown t, \varnothing) \in F \Rightarrow (s, \varnothing) \in F \qquad (N2)$$
$$(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F \qquad (N3)$$
$$(s, X) \in F \wedge (\forall\, c \in Y : (s^\frown\langle c\rangle, \varnothing) \notin F) \Rightarrow (s, X \cup Y) \in F \qquad (N4)$$
$$(\forall\, Y \in \mathbb{F}(X) : (s, Y) \in F) \Rightarrow (s, X) \in F \qquad (N5)$$

A.4 Semantics functions for CSP

Before defining the semantic functions $\mathcal{F}$ and $\mathcal{D}$ for CSP we give several preliminary definitions, that are used in the definition of $|||$, $_A\|_B$ and $\backslash$.

Definition A.1 The function merge applied to traces is defined as follows:

$$\mathsf{merge}: \Sigma^* \times \Sigma^* \to \mathbb{P}(\Sigma^*)$$

$$(i) \quad \mathsf{merge}(\langle\,\rangle, s) = \mathsf{merge}(s, \langle\,\rangle) = \{s\}$$
$$(ii) \quad \mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t) = \{\langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, \langle b\rangle^\frown t)\} \cup \{\langle b\rangle^\frown u \mid u \in \mathsf{merge}(\langle a\rangle^\frown s, t)\}$$

Definition A.2 The *restriction* function $\restriction$ is defined as follows:

$$\_ \restriction \_ : \Sigma^* \times \mathbb{P}(\Sigma) \to \Sigma^*$$

$$(i) \quad \langle\,\rangle \restriction X = \langle\,\rangle$$
$$(ii) \quad (\langle a\rangle^\frown s) \restriction X = \langle a\rangle^\frown(s \restriction X) \text{ – if } a \in X$$
$$= (s \restriction X) \text{ – if } a \notin X$$

Definition A.3 A version of the *hiding* operator which applies to *traces*, as opposed to processes ($s\backslash a$) is defined as follows:

$$\_\backslash\_ : \Sigma^* \times \mathbb{P}(\Sigma) \to \Sigma^*$$

$$(i) \quad \langle\,\rangle\backslash X = \langle\,\rangle$$
$$(ii) \quad (\langle a\rangle^\frown s)\backslash X = \langle a\rangle^\frown(s\backslash X) \text{ – if } a \notin X$$
$$= (s\backslash X) \text{ – if } a \in X$$

Definition A.4 The two projection functions failures and divergences select the respective components of the pair $\langle F, D\rangle$ as follows:

$$\mathsf{failures} : \mathbb{P}(\Sigma^* \times \mathbb{P}(\Sigma)) \times \mathbb{P}(\Sigma^*) \to \mathbb{P}(\Sigma^* \times \mathbb{P}(\Sigma))$$
$$\mathsf{divergences} : \mathbb{P}(\Sigma^* \times \mathbb{P}(\Sigma)) \times \mathbb{P}(\Sigma^*) \to \mathbb{P}(\Sigma^*)$$

$$(i) \quad \mathsf{failures}(\langle F, D\rangle) = F$$
$$(ii) \quad \mathsf{divergences}(\langle F, D\rangle) = D$$

A.4.1 Divergence sets

The divergence sets semantics functions are defined as follows:

$$
\begin{aligned}
\mathcal{D}[\![p]\!]e &= \mathsf{divergences}(e[\![p]\!]) \\
\mathcal{D}[\![\bot]\!]e &= \Sigma^* \\
\mathcal{D}[\![STOP]\!]e &= \varnothing \\
\mathcal{D}[\![SKIP]\!]e &= \varnothing \\
\mathcal{D}[\![a \rightarrow P]\!]e &= \{\, \langle a \rangle^\frown s \mid s \in \mathcal{D}[\![P]\!]e \,\} \\
\mathcal{D}[\![P \sqcap Q]\!]e &= \mathcal{D}[\![P]\!]e \cup \mathcal{D}[\![Q]\!]e \\
\mathcal{D}[\![P \,\Box\, Q]\!]e &= \mathcal{D}[\![P]\!]e \cup \mathcal{D}[\![Q]\!]e \\
\mathcal{D}[\![P \| Q]\!]e &= \{\, s^\frown t \mid s \in (\mathcal{D}[\![P]\!]e \cap \mathsf{traces}(Q)) \cup (\mathsf{traces}(P) \cap \mathcal{D}[\![Q]\!]e) \,\} \\
\mathcal{D}[\![P \,|||\, Q]\!]e &= \{\, u \mid \exists s, t : u \in \mathsf{merge}(s,t) \wedge (\quad (s \in \mathcal{D}[\![P]\!]e \wedge t \in \mathsf{traces}(Q)) \\
&\qquad\qquad \vee\ (s \in \mathsf{traces}(P) \wedge t \in \mathcal{D}[\![Q]\!]e)) \,\} \\
\mathcal{D}[\![P_A \|_B Q]\!]e &= \{\, u^\frown v \mid u \in (A \cup B)^* \wedge (\quad (u \upharpoonright A \in \mathcal{D}[\![P]\!]e \wedge u \upharpoonright B \in \mathsf{traces}(Q)) \\
&\qquad\qquad \vee\ (u \upharpoonright A \in \mathsf{traces}(P) \wedge u \upharpoonright B \in \mathcal{D}[\![Q]\!]e)) \,\} \\
\mathcal{D}[\![P; Q]\!]e &= \{ s^\frown t \mid \checkmark \text{ in } s \wedge s \in \mathcal{D}[\![P]\!]e \} \cup \{ s^\frown t \mid \checkmark \text{ in } s \wedge s^\frown\langle\checkmark\rangle \in \mathsf{traces}(P) \wedge t \in \mathcal{D}[\![Q]\!]e \} \\
\mathcal{D}[\![P \backslash a]\!]e &= \{ (s \backslash a)^\frown t \mid s \in \mathcal{D}[\![P]\!]e \} \cup \{ (s \backslash a)^\frown t \mid \forall n : s^\frown\langle a\rangle^n \in \mathsf{traces}(P) \} \\
\mathcal{D}[\![f^{-1}(P)]\!]e &= \{ s \mid f(s) \in \mathcal{D}[\![P]\!]e \} \\
\mathcal{D}[\![f(P)]\!]e &= \{ f(s)^\frown t \mid s \in \mathcal{D}[\![P]\!]e \}
\end{aligned}
$$

A.4.2 Failure sets

The failure sets semantics functions are defined as follows:

$$
\begin{aligned}
\mathcal{F}[\![p]\!]e &= \mathsf{failures}(e[\![p]\!]) \\
\mathcal{F}[\![\bot]\!]e &= \{ (s, X) \mid s \in \Sigma^* \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![STOP]\!]e &= \{ (\langle\,\rangle, X) \mid X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![SKIP]\!]e &= \{ (\langle\,\rangle, X) \mid \checkmark \notin X \wedge X \in \mathbb{P}(\Sigma) \} \cup \{ (\langle\checkmark\rangle, X) \mid X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![a \rightarrow P]\!]e &= \{ (\langle\,\rangle, X) \mid a \notin X \wedge X \in \mathbb{P}(\Sigma) \} \cup \{ (\langle a\rangle^\frown s, X) \mid (s, X) \in \mathcal{F}[\![P]\!]e \} \\
\mathcal{F}[\![P \sqcap Q]\!]e &= \mathcal{F}[\![P]\!]e \cup \mathcal{F}[\![Q]\!]e \\
\mathcal{F}[\![P \,\Box\, Q]\!]e &= \quad \{ (\langle\,\rangle, X) \mid (\langle\,\rangle, X) \in \mathcal{F}[\![P]\!]e \cap \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \neq \langle\,\rangle \wedge (s, X) \in \mathcal{F}[\![P]\!]e \cup \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P \,\Box\, Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![P \| Q]\!]e &= \quad \{ (s, X \cup Y) \mid (s, X) \in \mathcal{F}[\![P]\!]e \wedge (s, Y) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P \| Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![P \,|||\, Q]\!]e &= \quad \{ (u, X) \mid \exists s, t : u \in \mathsf{merge}(s,t) \wedge (s, X) \in \mathcal{F}[\![P]\!]e \wedge (t, X) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (u, X) \mid u \in \mathcal{D}[\![P \,|||\, Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![P_A \|_B Q]\!]e &= \quad \{ (u, X \cup Y \cup Z) \mid u \in (A \cup B)^* \wedge X \subseteq A \wedge Y \subseteq B \wedge Z \subseteq (\overline{A \cup B}) \\
&\qquad\qquad \wedge (u \upharpoonright A, X) \in \mathcal{F}[\![P]\!]e \wedge (u \upharpoonright B, Y) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (u, X) \mid u \in \mathcal{D}[\![P_A \|_B Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![P; Q]\!]e &= \quad \{ (s, X) \mid \checkmark \text{ in } s \wedge (s, X \cup \{\checkmark\}) \in \mathcal{F}[\![P]\!]e \} \\
&\quad \cup \{ (s^\frown t, X) \mid \checkmark \text{ in } s \wedge (s^\frown\langle\checkmark\rangle, \varnothing) \in \mathcal{F}[\![P]\!]e \wedge (t, X) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P; Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![P \backslash a]\!]e &= \{ (s \backslash a, X) \mid (s, X \cup \{a\}) \in \mathcal{F}[\![P]\!]e \} \cup \{ (s, X) \mid s \in \mathcal{D}[\![P \backslash a]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![f^{-1}(P)]\!]e &= \{ (s, X) \mid (f(s), f(X)) \in \mathcal{F}[\![P]\!]e \} \cup \{ (s, X) \mid s \in \mathcal{D}[\![f^{-1}(P)]\!]e \wedge X \in \mathbb{P}(\Sigma) \} \\
\mathcal{F}[\![f(P)]\!]e &= \{ (f(s), X) \mid (s, f^{-1}(X)) \in \mathcal{F}[\![P]\!]e \} \cup \{ (s, X) \mid s \in \mathcal{D}[\![f(P)]\!]e \wedge X \in \mathbb{P}(\Sigma) \}
\end{aligned}
$$

A.4.3 Failure-divergence sets for recursive processes

The denotation of a recursive process is built up in the usual way following the Fixed Point Theorem (see [Sto77]) for continuous functions. The semantic functions $\mathcal{D}$ and $\mathcal{F}$ for recursion in CSP are defined as follows:

$$\mathcal{N}[\![\mu p.P]\!]e \;=\; \mathsf{fix}(\lambda\langle F,\ D\rangle \bullet \mathcal{N}[\![P]\!](e \oplus [p \mapsto \langle F,\ D\rangle]))$$

$$\mathcal{D}[\![\mu p.P]\!]e \;=\; \bigcap_n \mathcal{D}[\![P_n]\!]e$$

$$\mathcal{F}[\![\mu p.P]\!]e \;=\; \bigcap_n \mathcal{F}[\![P_n]\!]e$$

where $P_0 = \bot$ and $P_{n+1} = P[p/P_n]$.

A.5 Semantic functions for the generalized and asynchronous termination parallel operators

A.5.1 Semantic functions for $P \|_\Omega Q$

The function used for combining the traces of P and Q to form those for $P\|_\Omega Q$ is merge. merge takes three arguments, two traces and a synchronization set, and maps them to the (possibly empty) set of traces for $P\|_\Omega Q$. If the value of $\mathsf{merge}(s, t, \Omega)$ is the empty set then it means that the two traces s and t are *incompatible*, because of synchronous events.

Definition A.5 The function $\mathsf{merge}(s, t, \Omega)$ is defined as follows:

$$\mathsf{merge} \;:\; \Sigma^* \times \Sigma^* \times \mathbb{P}(\Sigma) \;\to\; \mathbb{P}(\Sigma^*)$$

In the following $a \neq b$.

$$\begin{aligned}
&(i) && \mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) = \mathsf{merge}(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Omega)\\
&&&\qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, \langle b\rangle^\frown t, \Omega)\,\}\\
&&&\qquad\quad \cup \{\, \langle b\rangle^\frown u \mid u \in \mathsf{merge}(\langle a\rangle^\frown s, t, \Omega)\,\}\\
&&&\qquad\quad -\text{ if } a \notin \Omega,\ b \notin \Omega\\
&(ii) && \mathsf{merge}(\langle a\rangle^\frown s, \langle a\rangle^\frown t, \Omega) = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, t, \Omega)\,\}\\
&&&\qquad -\text{ if } a \in \Omega\\
&(iii) && \mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) = \mathsf{merge}(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Omega)\\
&&&\qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, \langle b\rangle^\frown t, \Omega)\,\}\\
&&&\qquad\quad -\text{ if } a \notin \Omega,\ b \in \Omega\\
&(iv) && \mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) = \varnothing\\
&&&\qquad -\text{ if } a \in \Omega,\ b \in \Omega\\
&(v) && \mathsf{merge}(\langle\,\rangle, \langle a\rangle^\frown s, \Omega) = \mathsf{merge}(\langle a\rangle^\frown s, \langle\,\rangle, \Omega)\\
&&&\qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}(\langle\,\rangle, s, \Omega)\,\}\\
&&&\qquad\quad -\text{ if } a \notin \Omega\\
&(vi) && \mathsf{merge}(\langle\,\rangle, \langle a\rangle^\frown s, \Omega) = \mathsf{merge}(\langle a\rangle^\frown s, \langle\,\rangle, \Omega)\\
&&&\qquad = \varnothing\\
&&&\qquad\quad -\text{ if } a \in \Omega\\
&(vii) && \mathsf{merge}(\langle\,\rangle, \langle\,\rangle, \Omega) = \{\, \langle\,\rangle\,\}
\end{aligned}$$

We now define the failure and divergence set semantic functions $\mathcal{F}$ and $\mathcal{D}$ for $P \|_\Omega Q$.

$$\begin{aligned}
\mathcal{F}[\![P \|_\Omega Q]\!]e = \quad & \{\, (u, X \cup Y \cup Z) \mid X \subseteq \overline{\Omega} \wedge Y, Z \subseteq \Omega \\
& \quad \wedge \exists s, t : u \in \mathsf{merge}(s, t, \Omega) \\
& \quad\quad \wedge (s, X \cup Y) \in \mathcal{F}[\![P]\!]e \wedge (t, X \cup Z) \in \mathcal{F}[\![Q]\!]e \,\} \\
& \cup \{\, (u, X) \mid u \in \mathcal{D}[\![P \|_\Omega Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \,\} \\
\mathcal{D}[\![P \|_\Omega Q]\!]e = \{\, & u^\frown v \mid \exists s, t : u \in \mathsf{merge}(s, t, \Omega) \\
& \quad \wedge (\quad (s \in \mathcal{D}[\![P]\!]e \wedge t \in \mathsf{traces}(Q)) \\
& \quad\quad \vee (s \in \mathsf{traces}(P) \wedge t \in \mathcal{D}[\![Q]\!]e)) \,\}
\end{aligned}$$

A.5.2 Semantic functions for $P /\!\!/_\Theta Q$

The definition of the semantic functions $\mathcal{F}$ and $\mathcal{D}$ for $P /\!\!/_\Theta Q$ are identical to those for $P \|_\Omega Q$ except that a different trace function is used instead of merge called merge✓. This new trace function reflects the fact that ✓ now has a special significance when occurring in a trace. The other difference between this operator and the generalized one is that ✓ is not allowed to be included in the synchronization set.

Definition A.6 The trace function $\mathsf{merge}\checkmark(s, t, \Theta)$ has the following signature:

$$\mathsf{merge}\checkmark \;:\; \Sigma^* \times \Sigma^* \times \mathbb{P}(\Sigma - \{\checkmark\}) \to \mathbb{P}(\{\, s, s^\frown\langle\checkmark\rangle \mid \checkmark \notin s \wedge s \in \Sigma^* \,\})$$

It is defined as follows, where throughout $a \neq b$, $a \neq \checkmark$ and $b \neq \checkmark$.

$$\begin{aligned}
(i) \quad & \mathsf{merge}\checkmark(\langle\,\rangle, \langle\,\rangle, \Theta) = \mathsf{merge}\checkmark(\langle\,\rangle, \langle\checkmark\rangle, \Theta) \\
& \qquad = \mathsf{merge}\checkmark(\langle\checkmark\rangle, \langle\,\rangle, \Theta) \\
& \qquad = \{\, \langle\,\rangle \,\} \\
(ii) \quad & \mathsf{merge}\checkmark(\langle\checkmark\rangle, \langle\checkmark\rangle, \Theta) = \{\, \langle\checkmark\rangle \,\} \\
(iii) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Theta) = \mathsf{merge}\checkmark(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \quad \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(s, \langle b\rangle^\frown t, \Theta) \,\} \\
& \qquad\quad \cup \{\, \langle b\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(\langle a\rangle^\frown s, t, \Theta) \,\} \\
& \qquad\quad - \text{if } a, b \notin \Theta \\
(iv) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle a\rangle^\frown t, \Theta) = \mathsf{merge}\checkmark(\langle a\rangle^\frown t, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(s, t, \Theta) \,\} \\
& \qquad\quad - \text{if } a \in \Theta \\
(v) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Theta) = \mathsf{merge}\checkmark(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \varnothing \\
& \qquad\quad - \text{if } a, b \in \Theta \\
(vi) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Theta) = \mathsf{merge}\checkmark(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(s, \langle b\rangle^\frown t, \Theta) \,\} \\
& \qquad\quad - \text{if } a \notin \Theta,\ b \in \Theta \\
(vii) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle\,\rangle, \Theta) = \mathsf{merge}\checkmark(\langle\,\rangle, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \varnothing \\
& \qquad\quad - \text{if } a \in \Theta \\
(viii) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle\checkmark\rangle, \Theta) = \mathsf{merge}\checkmark(\langle\checkmark\rangle, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \varnothing \\
& \qquad\quad - \text{if } a \in \Theta \\
(ix) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle\,\rangle, \Theta) = \mathsf{merge}\checkmark(\langle\,\rangle, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(s, \langle\,\rangle, \Theta) \,\} \\
& \qquad\quad - \text{if } a \notin \Theta \\
(x) \quad & \mathsf{merge}\checkmark(\langle a\rangle^\frown s, \langle\checkmark\rangle, \Theta) = \mathsf{merge}\checkmark(\langle\checkmark\rangle, \langle a\rangle^\frown s, \Theta) \\
& \qquad = \{\, \langle a\rangle^\frown u \mid u \in \mathsf{merge}\checkmark(s, \langle\checkmark\rangle, \Theta) \,\} \\
& \qquad\quad - \text{if } a \notin \Theta \\
(xi) \quad & \mathsf{merge}\checkmark(\langle\checkmark\rangle^\frown s, t, \Theta) = \mathsf{merge}\checkmark(t, \langle\checkmark\rangle^\frown s, \Theta) \\
& \qquad = \varnothing \\
& \qquad\quad - \text{if } s \neq \langle\,\rangle
\end{aligned}$$

The semantic functions $\mathcal{F}$ and $\mathcal{D}$ for the failure and divergence sets for $P /\!\!/_\Theta Q$ are defined as follows.

$$\begin{aligned}\mathcal{F}[\![P /\!\!/_\Theta Q]\!]e = \quad & \{(u, X \cup Y \cup Z) \mid X \subseteq \overline{\Theta} \wedge Y, Z \subseteq \Theta \\ & \wedge \exists s, t : u \in \mathsf{merge}\checkmark(s, t, \Theta) \\ & \wedge (s, X \cup Y) \in \mathcal{F}[\![P]\!]e \wedge (t, X \cup Z) \in \mathcal{F}[\![Q]\!]e\} \\ & \cup \{(u, X) \mid u \in \mathcal{D}[\![P /\!\!/_\Theta Q]\!]e \wedge X \in \mathbb{P}(\Sigma)\}\end{aligned}$$

$$\begin{aligned}\mathcal{D}[\![P /\!\!/_\Theta Q]\!]e = \{u^\frown v \mid & \exists s, t : u \in \mathsf{merge}\checkmark(s, t, \Theta) \\ & \wedge (\quad (s \in \mathcal{D}[\![P]\!]e \wedge t \in \mathsf{traces}(Q)) \\ & \quad \vee (s \in \mathsf{traces}(P) \wedge t \in \mathcal{D}[\![Q]\!]e))\}\end{aligned}$$

References

[AH92] Aceto L, Hennessy M (1992) Termination, deadlock, and divergence. J ACM 39(1):147–187

[BHR85] Brookes SD, Hoare CAR, Roscoe AW (1985) A theory of communicating sequential processes. J ACM 31(7)

[BR85] Brookes SD, Roscoe AW (1985) An improved failures model for communicating sequential processes. In: Proceedings of Pittsburgh Seminar on Concurrency, LNCS , vol 197. Springer, Heidelberg, pp 281–305

[Bro83] Brookes SD (1983) A model for communicating sequential processes. Ph.D. Thesis, Oxford University, Oxford

[BW90] Baeten JCM, Weijland WP (1990) Process algebra. Cambridge tracts in theoretical computer science, vol 18. Cambridge University Press, London

[CW06] Cavalcanti ALC, Woodcock JCP (2006) A tutorial introduction to CSP in unifying theories of programming. In: Refinement techniques in software engineering. Lecture Notes in Computer Science, vol. 3167. Springer, Heidelberg, pp 220–268

[Dav93] Davies J (1993) Specification and proof in Real-Time systems. Cambridge University Press, London

[DS89] Davies J, Schneider S (1989) An introduction to timed CSP. Technical monograph PRG-75, Programming Research Group. Oxford University, Oxford

[DS92] Davies J, Schneider S (1992) A brief history of timed CSP. Technical Monograph PRG-96, Programming Research Group, Oxford University, Oxford

[FDR03] Formal Systems (Europe) Ltd. Failure-divergence refinement: FDR2 Manual. Oxford, UK (2003)

[HJ98] Hoare CAR, He Jifeng (1998) Unifying Theories of Programming. Prentice-Hall, Englewood Cliffs

[Hoa85] Hoare CAR (1985) Communicating sequential processes. Prentice-Hall, Englewood Cliffs

[How05] Howells P (2005) Communicating sequential processes with flexibile parallel termination semantics. Ph.D. Thesis, University of Westminster

[Mil89] Milner R (1989) Communication and concurrency. Prentice-Hall, Englewood Cliffs

[Ree88] Reed GM (1988) A Uniform Mathematical Theory of Distributed Computing. Ph.D. thesis, Oxford University, Oxford

[Ree90] Reed GM (1990) A hierarchy of domains for Real-time distributed computing. In: Proceedings of 5th workshop on mathematical foundations of programming language semantics, LNCS 442. Springer, Heidelberg, pp 80–128

[Ros92] Roscoe AW (1992) An alternative order for the failures model. J Logic Comput 2(5):557–577

[Ros98] Roscoe AW (1998) The theory and practice of concurrency. Prentice-Hall, Englewood Cliffs

[RR96] Reed GM, Roscoe AW (1996) The timed failures-stability model for CSP. Technical monograph PRG-119, Programming Research Group, Oxford University, Oxford

[Sch90] Schneider S (1990) Correctness and communication in Real-time systems. Technical monograph PRG-84, Programming Research Group, Oxford University, Oxford

[Sto77] Stoy JE (1977) Denotational semantics: the Scott-Strachey approach to programming language theory. MIT Press, Cambridge

[TW97] Tej H, Wolff B (1997) 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 , vol 1313. Springer, Heidelberg

Received 26 October 2006

Accepted in revised form 30 October 2008 by J.C.P. Woodcock

Published online 6 December 2008