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

Specifying and Analysing Networks of Processes in CSP$_T$ (or In Search of Associativity)

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

Abstract. In proposing theories of how we should design and specify networks of processes it is necessary to show that the semantics of any language we use to write down the intended behaviours of a system has several qualities. First in that the meaning of what is written on the page reflects the intention of the designer; second that there are no unexpected behaviours that might arise in a specified system that are hidden from the unsuspecting specifier; and third that the intention for the design of the behaviour of a network of processes can be communicated clearly and intuitively to others. In order to achieve this we have developed a variant of CSP, called CSP$_T$, designed to solve the problems of termination of parallel processes present in the original formulation of CSP. In CSP$_T$ we introduced three parallel operators, each with a different kind of termination semantics, which we call synchronous, asynchronous and race. These operators provide specifiers with an expressive and flexible tool kit to define the intended behaviour of a system in such a way that unexpected or unwanted behaviours are guaranteed not to take place. In this paper we extend out analysis of CSP$_T$ and introduce the notion of an *alphabet diagram* that illustrates the different categories of events that can arise in the parallel composition of processes. These alphabet diagrams are then used to analyse networks of three processes in parallel with the aim of identifying sufficient constraints to ensure associativity of their parallel composition. Having achieved this we then proceed to prove associativity laws for the three parallel operators of CSP$_T$. Next, we illustrate how to design and construct a network of three processes that satisfy the associativity law, using the associativity theorem and alphabet diagrams. Finally, we outline how this could be achieved for more general networks of processes.

Keywords. concurrency, CSP, CSP$_T$, parallel operators, associativity, process networks

Introduction

In the original *failure-divergence* semantic model for *Communicating Sequential Processes* (CSP) developed by Hoare, Brookes and Rosoce [1,2,3,4] the incomplete treatment of successful process termination, and in particular parallel termination, permitted intuitively contradictory processes to be defined. For example, it was possible to define a parallel process that appeared to have terminated several times before actually doing so. In many cases this meant that the behaviour of the system did not match the intention of the specifier when it was specified formally using CSP.

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

In response to this problem we developed a variant of CSP as defined in the original work of Hoare and its development by Roscoe [5,6], called CSP$_T$ that has been introduced elsewhere [7,8]. In this work we demonstrated how CSP$_T$, solves the original termination problems by introducing three distinct but related parallel operators that between them provide a transparent and intuitive means for specifying the desired termination of networks of processes. These three parallel operators each have a different form of termination semantics based on different ways of combining the termination of its component processes. The three forms are *synchronous* termination, *asynchronous* termination and *race* termination (where the first component process to terminate, terminates the composite process).

The design of these three parallel operators, was motivated by the desire to provide a system designer with both the *precision* and *flexibility* to be able to select as wide a range of different kinds of termination and interaction for networks of parallel processes. In doing so we had two overarching principles: (i) to design a semantic model as consistent as possible with the original model defined for CSP; and (ii) to design a model that provided any system designer with the confidence that the system would behave as intended.

By providing these three operators, we claim (and indeed have proved) that any system specifier can now make precise choices about how parallel networks of processes are required to successfully terminate and interact by means of synchronised events.

The need for such parallel operators can be seen when we consider the range of concurrent systems, with different termination behaviour, that we would wish to specify and design. For example, (distributed) client/server systems usually require the asynchronous termination of the client and server processes. In contrast, for a system of hardware components, synchronous termination would normally be a fundamental requirement. For parallel searching, race termination would be desirable, since, it is only necessary to wait for the first (and possibly only) successful process to terminate. By providing parallel operators with different termination semantics, a more natural way of modelling and designing modern parallel systems in different environments becomes possible.

In [8], we focused on the specification of how parallel processes should terminate using CSP$_T$'s three parallel operators. In this paper we continue the investigation of CSP$_T$'s parallel operators, by focusing on the specification of parallel process interaction. In particular, the different types of synchronous and asynchronous events that can occur in a parallel composition. A key aspect of this investigation is the discovery of constraints for and proof of associativity laws for the three parallel operators. These laws provide specifiers and designers with an essential law for developing, analysing and reasoning about their specifications and designs of networks of processes. For example, as we will demonstrate, they allow us to construct a process network using the parallel operators based on the types of interactions required. Further, the associativity constraints can be used to determine if the associativity law applies for a given process network and determine if a network with specific types of interactions is constructable.

Aims and Motivations

The main goal of the paper is to prove as general an associativity law as possible for the generalised parallel operator of CSP and of our three operators - race, synchronous and asynchronous of CSP$_T$. As Roscoe [6] acknowledges the generalised parallel operator for CSP has become the most widely used and so there is a need for as strong an associativity law as possible to enable CSP practitioners to design and analyse parallel networks of processes. Roscoe has introduced a weak notion of associativity and we have improved the strength of the associativity law. The weak law, for example, provides little practical use when designing and analysing networks of parallel processes that are constructed using it. Therefore, there is a clear need for a strengthened version of the associativity law. Of course Roscoe does pro-

vide a strong associativity law for the alphabetised parallel operator (see Section 3.2). However, this law only deals with the case when the synchronising alphabets of the operator are the same as the alphabets of the process operands. Clearly this is limiting for any specifier.

Furthermore, one of our aims is to make the analysis and design of CSP process networks more tractable for system developers. To date there appears to be no methodology specifically aimed at designing and analysing the interactions in these networks. In this paper we attempt to address this by presenting a detailed analysis and categorisation of the event types that can occur in these types of networks. When this is combined with the use of alphabet diagrams, it provides a simple methodology to help developers design and understand the interactions between processes in these networks.

Outline of the Rest of the Paper

In Section 1, we provide a brief introduction to CSP$_T$. *Alphabet diagrams* are introduced in Section 2. These illustrate the different categories of events, and thus interactions, that can arise in the parallel composition of processes. In Section 3, we use alphabet diagrams to identifying constraints sufficient to ensure associativity laws for the three parallel operators of CSP$_T$. We then illustrate, in Section 4, how to use the associativity law and alphabet diagrams to design and construct a simple network of process that satisfy the associativity law. We suggest further work, in Section 5. Finally, we present our conclusions in Section 6. (Appendix A contains a summary of the notation, definitions and functions used in the paper. Appendix B details the proof of the main Associativity result presented in Section 3.5.)

1. An Introduction to CSP$_T$[1]

In this section[2],we provide an overview of CSP$_T$, for a complete description please see [7, 8,10]. (For a summary of the CSP and CSP$_T$ notation and definitions used throughout the paper, see Appendix A.)

In Hoare, Brookes and Roscoe's original CSP [1,2,3,4] there was a well known problem concerning the incomplete treatment of parallel termination by two of original parallel operators: alphabetised ($_A\|_B$)[3] and asynchronous/interleaving ($|||$). In particular, the type of termination that occurred using these operators was inconsistent and could vary depending on the termination of the processes being combined.

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

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

Clearly the $\checkmark$s on the right hand side cannot be interpreted as the successful termination of the process $(a \rightarrow SKIP) ||| (b \rightarrow SKIP)$, since it continues to perform a, b and $\checkmark$ events. This illustrates how the original semantics for CSP could gave rise to these obviously undesirable and intuitively contradictory processes.

[1]This section contains material extracted from Sections 3.1, 5.2 and 5.6 of [7] and Sections 3.2, 3.3 and 3.4 of [8] by permission of Springer and Elsevier, respectively.

[2]This section is identical to Section 1 of [9], a companion paper in these same Proceedings (CPA 2013). This duplication, by permission of the Editors, is to let both papers be self-contained.

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

Several solutions have been proposed to this problem, including Tej and Wolf [11], Roscoe [5,6], Hoare and He [12] and the authors's own solution CSP_T [7]. For a detailed comparison of these solutions the interested reader is referred to [7].

Our starting point in defining CSP_T was the original failure-divergence model developed by Hoare, Brookes and Roscoe [3]. Our aim in modifying this model was to provide a more robust treatment of termination through the consistent and special handling of ✓ by the language (processes and operators) and semantics (failures and divergences).

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

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

where s and t are traces, F and D are the failure and divergence sets respectively of a process. This axiom means that if a process indicates that it has terminated (by means of the ✓) but continues to perform events (t), then it must have started diverging before it performed the ✓ (i.e. $s \in D$). For the rational behind this axiom, see our companion paper [9].

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

1.1. Parallel Operators of CSP_T

Our three new parallel operators are defined to be used as replacements for the original synchronous ($\|$), interleaving ($|||$) and alphabetised ($_A\|_B$) parallel operators. It is necessary to define replacements for $|||$ and $_A\|_B$, as they do not satisfy (T1). These new parallel operators are *generalised* (or *interface*) style parallel operators, i.e. are parameterised by the set of events the processes are required to synchronise on. Each of these three operators has a distinct type of parallel termination semantics, and thus are distinct operators, see [8] for details. We call them *synchronous*, *asynchronous* and *race*, that we define here.

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

Asynchronous: requires the successful termination of both P and Q; and P and Q terminated asynchronously, i.e. they do not synchronise on ✓. (Roscoe [5,6] refers to this type of parallel termination semantics as *distributed termination*.)

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

Note that in [5,6] Roscoe chooses to reject race termination semantics, due to the problems of dealing with the non-terminated process, e.g. the need for some powerful mechanism to manage its termination. We believe, however, that it is preferable at least to offer system specifiers the choice of using this type of termination semantics and let them resolve these issues, rather than ban it outright. We believe that this is not against the spirit of CSP, since

there are other features available in CSP that also raise similar implementation issues, e.g. the various interrupt style operators.

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

Table 1. CSP$_T$ parallel operators

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

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

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

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

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

Here are examples of the three parallel operators of CSP$_T$, that illustrate the difference between $|_\Theta$ and the two others (which for these examples, with empty synchronisation sets, are the same):

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

It is important to note that we do not include the generalised parallel operator $\|_\Omega$ in the language of CSP$_T$, since this would defeat the purpose of the whole exercise. Since, it would again result in inconsistent processes, similar to the one given above in Section 1, with multiple ticks in its traces. However, by restricting its use to defining $\|_\Delta$ and $|_\Theta$ we ensure these forms of processes do not occur. The semantic functions for the generalised operator are given in Appendix A. Full details for the other processes and operators, including the operational semantics and all semantic functions, can be found in [7,8].

1.2. The Model and Language of CSP$_T$

The model for CSP$_T$, N_T is defined by adding the Termination axiom (T1) to the process axioms (D1) to (N5) of the original CSP model N, given in Appendix A. Hence, in N_T the failure and divergence sets of a process are defined as for N, except that they also satisfy the process axiom (T1) as well as (D1) to (N5). The semantic functions $\mathcal{F}$ and $\mathcal{D}$ for N_T are the same as for N, and are given in [7].

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

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

where $a \in \Sigma - \{\checkmark\}$. $\bot$ is the divergent process (can be defined as $\mu p.p$). *STOP* is the deadlocked process and *SKIP* is the successfully terminating process. $a \rightarrow P$ is action prefix. $P \sqcap Q$ is nondeterministic choice and $P \,\Box\, Q$ is deterministic choice. $P; Q$ is sequential composition. $P\backslash a$ is event hiding. $P[\![R]\!]$ is action (relational) renaming, with the usual constraint [5,6] applying to the renaming relation R with respect to $\checkmark$, i.e. that no other event is mapped to it or that it is mapped to another event. p is a process variable, $\mu p.F(p)$ is recursion and in the definition of $F(p)$, only the above processes and operators can be used. $P\|_\Delta Q$, $P|||_\Theta Q$ and $P|_\Theta Q$ are the generalised synchronous, asynchronous and race parallel operators respectively. The processes and operators of CSP$_T$ are *well-defined* and *well-behaved* in N_T.

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

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

We can now build on the account given so far in order to begin to investigate associativity of networks of CSP$_T$ processes in the following sections.

2. Alphabet Diagrams

We introduce the notion of an *alphabet diagram*, which is a method of analysing parallel composition by means of the types of events that could occur or could not occur in its execution. An *alphabet diagram* is simply a Venn diagram that allows us to consider the different types of events that are involved in a parallel process throughout its lifetime. These events are subdivided into *synchronous* and *asynchronous* events. This subdivision allows us to characterise a parallel process depending on the presence or absence of these types of events. In [13], Schneider uses a related form of event diagram for parallel composition, but only to illustrate what events are initially possible.

As an example, consider the generalised parallel composition $P\|_\Omega Q$ of two processes P and Q, with alphabets A and B respectively, that are required to synchronise on a particular set of events Ω. The semantics of $P\|_\Omega Q$ is that P and Q are composed in parallel and are required to synchronise 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. The event types for $P\|_\Omega Q$ are illustrated in the alphabet diagram of Figure 1 and defined below.

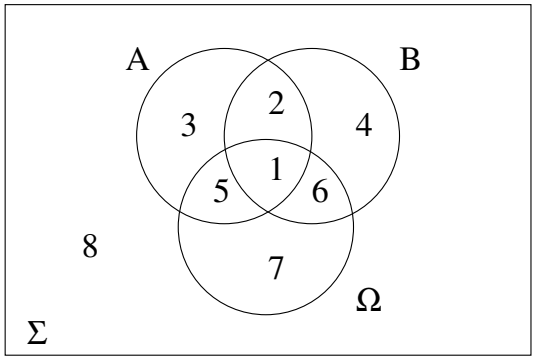

Figure 1. Alphabet diagram for $P\|_\Omega Q$.

1. *Possible synchronous* events $(A \cap B \cap \Omega)$: P and Q could synchronise on these since they are possible for both of them. Events of this type can only occur when both processes agree. They are called *possible* synchronous events because P and Q may never be willing to perform them at the same time, hence, they may never be performed.
2. *Common asynchronous* events $(A \cap B \cap \overline{\Omega})$: P and Q do not synchronise on these, even though they are possible for both of them. This type of event can be performed by either P or Q.
3. *P's private asynchronous* events $(A \cap \overline{B} \cap \overline{\Omega})$: only possible for P and since they are not in the synchronisation set they are performed independently of Q.
4. *Q's private* asynchronous events $(\overline{A} \cap B \cap \overline{\Omega})$: as for *P's*.
5. *P's inhibited synchronous* events $(A \cap \overline{B} \cap \Omega)$: only possible for P but they are also in the synchronisation set. Therefore, they require the participation of Q, which is impossible as they are not in Q's alphabet, hence, these events cannot be performed.
6. *Q's inhibited synchronous* events $(\overline{A} \cap B \cap \Omega)$: as for *P's*.
7. *Irrelevant synchronous* events $(\overline{A} \cap \overline{B} \cap \Omega)$: not in the alphabets of either process, and consequently cannot be performed.
8. *Irrelevant* events $(\overline{A} \cap \overline{B} \cap \overline{\Omega})$: not in the alphabets of either of the two processes nor are they required to be synchronised on.

Clearly *Irrelevant synchronous* events and *Irrelevant* events can never occur in the trace of any parallel process composed of P and Q because they are not in their alphabets. We therefore, only need to consider events which are actually possible for the two processes, i.e. within the union of their alphabets $A \cup B$, regions 1 to 6.

The eight types of events are the most which can occur for any given P, Q and Ω using this operator. However, for a particular instance of $P\|_\Omega Q$ only a subset of these types of events may be present. The presence or absence of these event types in a given instance of $P\|_\Omega Q$ can be used to classify different types of parallel composition.

3. Associativity of the Parallel Operators

In our search for an associativity law for each of the CSP$_T$'s parallel operators we concentrate on finding an associativity law for the generalised parallel operator – $\|_\Omega$. The reason for this is that it is used to define two of the three operators: synchronous ($\|_\Delta$) and race ($|_\Theta$). Further more, any associativity law for this operator will form the basis for the associativity law for the asynchronous operator ($|||_\Theta$). In addition, for the purposes of this present investigation we treat tick ($\checkmark$) as any other event that a process could perform.

3.1. Associativity of the Generalised Operator

The most important property required of $P\|_\Omega Q$ after it has been shown to be a well-defined and well-behaved process is *associativity*. That is, for what values of Λ_1, Λ_2, Π_1, Π_2, Γ_1 and Γ_2 does the following hold?

$$P\|_{\Lambda_1}(Q\|_{\Lambda_2}R) \;\equiv\; Q\|_{\Pi_1}(P\|_{\Pi_2}R) \;\equiv\; (P\|_{\Gamma_1}Q)\|_{\Gamma_2}R$$

We will refer to each of these processes as the (Λ), (Π) and (Γ) versions respectively.

It is obvious that constraints must be placed on the synchronisation sets, since combining processes using the existing parallel operators either in a different order or with different operators are rarely equivalent; that is, in general none of the following hold:

$$\begin{aligned}
P\|(Q|||R) &\equiv (P\|Q)|||R \\
P|||(Q\|R) &\equiv (P|||Q)\|R \\
P|||(Q_B\|_C R) &\equiv (P|||Q)_{A\cup B}\|_C R \\
P|||(Q_B\|_C R) &\equiv (P\|Q)\|R
\end{aligned}$$

3.2. Related Work

Roscoe [6] page 60, gives the following "weak (in that both interfaces are the same)" associativity law for his generalised parallel operator (that has asynchronous termination semantics).

$$P\|_X(Q\|_X R) \;=\; (P\|_X Q)\|_X R \qquad \langle \|_X\text{−assoc}\rangle$$

He states that it is difficult to "... construct a universally applicable and elegant associativity law.", for this type of operator due to the various types of events that can occur. In this context "universal" means that the synchronisation set X is independent of the alphabets of the three processes. He gives as an example, the process $P\|_X(Q\|_Y R)$ and an event that could occur in X but not in Y that both Q and R can perform. In what follows we shall denote this type of event as a *synchronous common* event, and denote it by *(QRa)Ps*, for processes similar to Roscoe's example. In addition, we shall show that if an event of this type is present in these types of processes then they cannot be transformed using a "universally applicable" associativity law. Schneider [13] gives a similar universal associativity law, for his "interface parallel" operator which has synchronous termination semantics.

Roscoe [6] gives an associativity law for his alphabetised parallel operator (with asynchronous termination) and gives an equivalence relating it to his generalised one as follows:

$$\begin{aligned}
(P_A\|_B Q)_{A\cup B}\|_C R \;&=\; P_A\|_{B\cup C}\,(Q_B\|_C R) \qquad \langle {}_A\|_B\text{−assoc}\rangle \\
(P_A\|_B Q) \;&=\; P\|_{A\cap B}\,Q
\end{aligned}$$

where A, B and C are $\alpha(P)$, $\alpha(Q)$ and $\alpha(R)$ respectively. By combining these two, it is possible to produce a less universal but stronger associativity law for the generalised operator than $\langle \|_X\text{−assoc}\rangle$. This new law would be applicable when all events in the intersections of the processes' alphabets are synchronised. However, it does not cover the cases when any common asynchronous events are present, for example, when an event in $A \cap B$ is required to be asynchronous, because by definition of ${}_A\|_B$ it must be synchronous.

Our aim in Section 3 is, therefore, to discover as general an associativity law as possible for our generalised parallel operator. Such laws are relevant to standard CSP because they capture intrinsic properties about parallel associativity and thus can be translated directly into associativity laws for the parallel operators of standard CSP.

3.3. Event Types for Three Processes

To identify the constraints on the synchronisation sets that are required to make the operator associative, we shall consider all of the possible event types that can occur when three processes are combined.

We have seen that when two processes are combined there are eight types of events but only six of these are relevant. When three are combined there are thirty two, of which twenty eight are relevant. Some of these are new types of events, others are just a natural extension of the existing types when three processes are combined. Below are listed all of the types of events that can occur, with a definition of the new types of events.

Private asynchronous events: are only performed by a single process, denoted for each process by – *Pa*, *Qa*, *Ra*.

Possible binary synchronous events: pairs of processes can synchronise on these. This excludes other forms of synchronous events such as ternary synchronous events etc. Denoted for each pair of processes by – *PQs*, *PRs*, *QRs*.

Common binary asynchronous events: are restricted to only two processes. Denoted for each pair of processes by – *PQa*, *PRa*, *QRa*.

Inhibited events: are due to the first synchronisation set that has effect on the process. Denoted for each process by – *Pi*, *Qi*, *Ri*. For example, in $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$, *Pi* events are due to Λ_1, *Qi* and *Ri* events are due to Λ_2.

Inhibited private events: are private asynchronous events under the first synchronisation set but are then inhibited by the second synchronisation set which has effect on the process. Denoted for each process by – *(Pa)i*, *(Qa)i*, *(Ra)i*. For example, in $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$ only *(Qa)i* and *(Ra)i* events are present, they are not in Λ_2 but are in Λ_1. No *(Pa)i* events are present since only one synchronisation set affects *P*.

Possible ternary synchronous events: all three processes are required to synchronise on these. Therefore, they are included in both synchronisation sets. Denoted by – *PQRs*.

Common ternary asynchronous events: all three processes can perform these asynchronously, hence, they are not included in either synchronisation set. Denoted by – *PQRa*.

Common synchronous events: are possible synchronous events because of the first synchronisation set but then become common asynchronous events with the third process. Denoted by – *(PQs)Ra*, *(PRs)Qa*, *(QRs)Pa*. Note that only one of these types of events can occur in any one of the three processes, for example, in $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$ only *(QRs)Pa* events can occur.

Synchronous common events: are common asynchronous events under the first synchronisation set but then become possible synchronous events when combined with the third process. Denoted by – *(PQa)Rs*, *(PRa)Qs*, *(QRa)Ps*. Again only one of these types of event can be present in either of the three processes, for example, in $Q\|_{\Pi_1}(P\|_{\Pi_2}R)$ only *(PRa)Qs* events can occur.

Inhibited synchronous events: are possible synchronous events because of the first synchronisation set but are then required to be possible synchronous events with the third process which cannot perform them and are therefore, inhibited. Denoted by – *(PQs)i*, *(PRs)i*, *(QRs)i*. Only one of these event types can be present in each of the three processes, for example, *(QRs)i* in $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$.

Inhibited common events: are common asynchronous events under the first synchronisation set but are then required to be possible synchronous events with the third process which cannot perform them and are therefore, inhibited. Denoted by – *(PQa)i*, *(PRa)i*, *(QRa)i*. Only one of these event types can be present, for example, *(PRa)i* in $Q\|_{\Pi_1}(P\|_{\Pi_2}R)$.

Irrelevant synchronous events: are not in the alphabets of the two processes being combined, and consequently have no effect on the resultant process. Denoted by – *PQis*, *PRis*, *QRis*.

Irrelevant events: are outside the alphabets of all three processes and the synchronisation sets involved, denoted by – *PQRi*.

3.3.1. Alphabet Diagram for Three Processes

Only certain combinations of these events can occur in one of the ways of combining *P*, *Q* and *R*. To see what part of the three processes' alphabets form each event type for each of the different combinations we give one alphabet diagram, see Figure 2, and use it to represent each of the three processes one at a time. That is S_1 and S_2 represent Λ_1, Λ_2, Π_1, Π_2, Γ_1 and Γ_2 respectively.

Note that even though a particular area is (visually) the same in the diagram under all three interpretations, this is because the same two groups of areas are used to represent the three pairs of synchronisation sets. The areas are not intended to be equal, as would be seen if four other groups of areas were introduced to represent the other pairs of synchronisation sets.

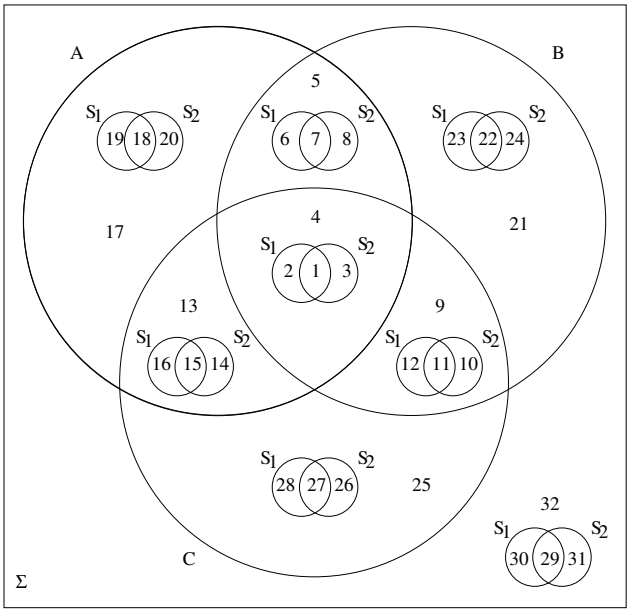

Figure 2. Alphabet diagram for three processes.

3.3.2. Tabulation of Event Types

Below in Tables 3 – 9 are tabulated the areas of the alphabet diagram of Figure 2, which comprise each of the event types for each of the three processes. Each area corresponds to a distinct region of the alphabet diagram, indicated by the number associated with it. If an event type is not possible then its area is given as the empty set. Due to limitations of space we have not included the tables for the various types of inhibited and irrelevant events[4]. We use the additional notation of Γ, Λ and Π as superscripts of the event types to indicate which of the three processes are under consideration, e.g. Pa^{Γ} represents the private asynchronous events of *P* in the Γ process.

[4]An extended version of this paper is in preparation and the full details of the inhibited and irrelevant events will be included there.

Table 3. Private asynchronous events: *Pa*, *Qa*, *Ra*

Event	Set of Events	*Area*
Pa^{Γ}	$(A \cap \overline{B} \cap \overline{C} \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	17
Pa^{Λ}	$(A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	20, 17,
	$\cup (A \cap B \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap C \cap \overline{\Lambda_1} \cap \Lambda_2)$	8, 14
Pa^{Π}	$(A \cap \overline{B} \cap \overline{C} \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	17
Qa^{Γ}	$(\overline{A} \cap B \cap \overline{C} \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	21
Qa^{Λ}	$(\overline{A} \cap B \cap \overline{C} \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	21
Qa^{Π}	$(A \cap B \cap \overline{C} \cap \overline{\Pi_1} \cap \Pi_2) \cup (\overline{A} \cap B \cap C \cap \overline{\Pi_1} \cap \Pi_2)$	8, 10,
	$\cup (\overline{A} \cap B \cap \overline{C} \cap \overline{\Pi_1} \cap \Pi_2) \cup (\overline{A} \cap B \cap \overline{C} \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	24, 21
Ra^{Γ}	$(\overline{A} \cap B \cap C \cap \Gamma_1 \cap \overline{\Gamma_2}) \cup (\overline{A} \cap \overline{B} \cap C \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	12, 25
	$\cup (\overline{A} \cap \overline{B} \cap C \cap \Gamma_1 \cap \overline{\Gamma_2}) \cup (A \cap \overline{B} \cap C \cap \Gamma_1 \cap \overline{\Gamma_2})$	28, 16
Ra^{Λ}	$(\overline{A} \cap \overline{B} \cap C \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	25
Ra^{Π}	$(\overline{A} \cap \overline{B} \cap C \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	25

Table 4. Common binary asynchronous events: *PQa*, *PRa*, *QRa*

Event	Set of Events	*Area*
PQa^{Γ}	$(A \cap B \cap \overline{C} \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	5
PQa^{Λ}	$(A \cap B \cap \overline{C} \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	5
PQa^{Π}	$(A \cap B \cap \overline{C} \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	5
PRa^{Γ}	$(A \cap \overline{B} \cap C \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	13
PRa^{Λ}	$(A \cap \overline{B} \cap C \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	13
PRa^{Π}	$(A \cap \overline{B} \cap C \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	13
QRa^{Γ}	$(\overline{A} \cap B \cap C \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	9
QRa^{Λ}	$(\overline{A} \cap B \cap C \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	9
QRa^{Π}	$(\overline{A} \cap B \cap C \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	9

Table 5. Common ternary asynchronous events: *PQRa*

Event	Set of Events	*Area*
$PQRa^{\Gamma}$	$(A \cap B \cap C \cap \overline{\Gamma_1} \cap \overline{\Gamma_2})$	4
$PQRa^{\Lambda}$	$(A \cap B \cap C \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})$	4
$PQRa^{\Pi}$	$(A \cap B \cap C \cap \overline{\Pi_1} \cap \overline{\Pi_2})$	4

3.4. Defining Associative Synchronisation Sets

For associativity we require the three parallel alternatives to be equivalent. For this to be the case a necessary condition is firstly that all three processes have the same event types present and secondly that each event type contains the same set of events in each process. We shall apply this constraint only to the event types that could actually occur and not to any particular type of inhibited events, because in general it is unimportant how an event is inhibited.

So from Table 3 it is clear that the *Pa*, *Qa* and *Ra* must be restricted in some way. Consider *Pa*, it contains events which are present in the (Λ) process which are not of the same type in the other two processes, i.e. areas 8, 14 and 20. Therefore, as a first step we require the following to hold:

Table 6. Possible binary synchronous events: *PQs*, *PRs*, *QRs*

Event	Set of Events	*Area*
PQs^{Γ}	$(A \cap B \cap \overline{C} \cap \Gamma_1 \cap \overline{\Gamma_2})$	6
PQs^{Λ}	$(A \cap B \cap \overline{C} \cap \Lambda_1 \cap \overline{\Lambda_2})$	6
PQs^{Π}	$(A \cap B \cap \overline{C} \cap \Pi_1 \cap \overline{\Pi_2})$	6
PRs^{Γ}	$(A \cap \overline{B} \cap C \cap \overline{\Gamma_1} \cap \Gamma_2)$	14
PRs^{Λ}	$(A \cap \overline{B} \cap C \cap \Lambda_1 \cap \overline{\Lambda_2})$	16
PRs^{Π}	$(A \cap \overline{B} \cap C \cap \overline{\Pi_1} \cap \Pi_2)$	14
QRs^{Γ}	$(\overline{A} \cap B \cap C \cap \overline{\Gamma_1} \cap \Gamma_2)$	10
QRs^{Λ}	$(\overline{A} \cap B \cap C \cap \overline{\Lambda_1} \cap \Lambda_2)$	10
QRs^{Π}	$(\overline{A} \cap B \cap C \cap \Pi_1 \cap \overline{\Pi_2})$	12

Table 7. Possible ternary synchronous events: *PQRs*

Event	Set of Events	*Area*
$PQRs^{\Gamma}$	$(A \cap B \cap C \cap \Gamma_1 \cap \Gamma_2)$	1
$PQRs^{\Lambda}$	$(A \cap B \cap C \cap \Lambda_1 \cap \Lambda_2)$	1
$PQRs^{\Pi}$	$(A \cap B \cap C \cap \Pi_1 \cap \Pi_2)$	1

Table 8. Common synchronous events: *(PQs)Ra*, *(PRs)Qa*, *(QRs)Pa*

Event	Set of Events	*Area*
$(PQa)Rs^{\Gamma}$	$(A \cap B \cap C \cap \overline{\Gamma_1} \cap \Gamma_2)$	3
$(PQa)Rs^{\Lambda}$	$\varnothing$	
$(PQa)Rs^{\Pi}$	$\varnothing$	
$(PRa)Qs^{\Gamma}$	$\varnothing$	
$(PRa)Qs^{\Lambda}$	$\varnothing$	
$(PRa)Qs^{\Pi}$	$(A \cap B \cap C \cap \Pi_1 \cap \overline{\Pi_2})$	2
$(QRa)Ps^{\Gamma}$	$\varnothing$	
$(QRa)Ps^{\Lambda}$	$(A \cap B \cap C \cap \Lambda_1 \cap \overline{\Lambda_2})$	2
$(QRa)Ps^{\Pi}$	$\varnothing$	

$$
\begin{aligned}
Pa &= (A \cap \overline{B} \cap \overline{C} \cap \overline{\Gamma_1} \cap \overline{\Gamma_2}) \\
&= \quad (A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \overline{\Lambda_2}) \\
&\quad \cup\ (A \cap B \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap C \cap \overline{\Lambda_1} \cap \overline{\Lambda_2}) \\
&= (A \cap \overline{B} \cap \overline{C} \cap \overline{\Pi_1} \cap \overline{\Pi_2})
\end{aligned}
$$

As a second step it is necessary to eliminate those events which are present in the (Λ) process which do not form part of *Pa* in the others. Therefore, the following equalities must hold:

$$
\begin{aligned}
Pa &= (A \cap \overline{B} \cap \overline{C} \cap \overline{\Gamma_1} \cap \overline{\Gamma_2}) \\
&= (A \cap \overline{B} \cap \overline{C} \cap \overline{\Pi_1} \cap \overline{\Pi_2}) \\
&= (A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap \overline{C} \cap \overline{\Lambda_1} \cap \overline{\Lambda_2})
\end{aligned}
$$

Table 9. Synchronous common events: *(PQa)Rs*, *(PRa)Qs*, *(QRa)Ps*

Event	Set of Events	*Area*
(PQs)Ra$^{\Gamma}$	$(A \cap B \cap C \cap \Gamma_1 \cap \overline{\Gamma_2})$	2
(PQs)Ra$^{\Lambda}$	$\varnothing$	
(PQs)Ra$^{\Pi}$	$\varnothing$	
(PRs)Qa$^{\Gamma}$	$\varnothing$	
(PRs)Qa$^{\Lambda}$	$\varnothing$	
(PRs)Qa$^{\Pi}$	$(A \cap B \cap C \cap \overline{\Pi_1} \cap \Pi_2)$	3
(QRs)Pa$^{\Gamma}$	$\varnothing$	
(QRs)Pa$^{\Lambda}$	$(A \cap B \cap C \cap \overline{\Lambda_1} \cap \Lambda_2)$	3
(QRs)Pa$^{\Pi}$	$\varnothing$	

In addition, the following equality must hold, because there is no way that the *Pa* events of the (Γ) and (Π) versions could contain any events from the $A \cap B \cap \overline{C}$ and $A \cap \overline{B} \cap C$ areas.

$$(A \cap B \cap \overline{C} \cap \overline{\Lambda_1} \cap \Lambda_2) \cup (A \cap \overline{B} \cap C \cap \overline{\Lambda_1} \cap \Lambda_2) = \varnothing$$

Two possible restrictions that satisfy these equalities are the following:

$$A \cap \Lambda_2 = \varnothing \qquad \text{and} \qquad A \cap \overline{\Lambda_1} \cap \Lambda_2 = \varnothing$$

The problem with the first is that it would also eliminate the possibility of *PQRs* events, for this reason we choose the second. This leads to similar constraints for Γ_1, Γ_2, Π_1 and Π_2 derived from *Qa* and *Ra* as follows:

$$B \cap \overline{\Pi_1} \cap \Pi_2 = \varnothing \qquad\qquad C \cap \Gamma_1 \cap \overline{\Gamma_2} = \varnothing$$

From Table 8 it is clear that none of the events: *(PQa)Rs*, *(PRa)Qs* and *(QRa)Ps*, should occur in either of the three alternatives. Since there is no way in which they can occur in all of them. Therefore, constraints must be placed on Γ_1, Γ_2, Λ_1, Λ_2, Π_1 and Π_2 so that they are eliminated. If these events are eliminated then so are the corresponding inhibited ones: *(PQa)i*, *(PRa)i* and *(QRa)i*.

Similarly, from Table 9 we must eliminate: *(PQs)Ra*, *(PRs)Qa* and *(QRs)Pa*. However, the constraints that were introduced to eliminate the problems involving *Pa*, *Qa* and *Ra* also eliminate these events, as can be seen from Table 9.

Therefore, we must only find constraints to eliminate *(PQa)Rs*, *(PRa)Qs* and *(QRa)Ps*. So we require the following equalities to hold:

$$\begin{aligned} \textit{(QRa)Ps} &= (A \cap B \cap C \cap \Lambda_1 \cap \overline{\Lambda_2}) = \varnothing \\ \textit{(PQa)Rs} &= (A \cap B \cap C \cap \overline{\Gamma_1} \cap \Gamma_2) = \varnothing \\ \textit{(PRa)Qs} &= (A \cap B \cap C \cap \Pi_1 \cap \overline{\Pi_2}) = \varnothing \end{aligned}$$

Just considering *(QRa)Ps* three possible restrictions which satisfy this equality are the following:

$$B \cap \Lambda_1 \cap \overline{\Lambda_2} = \varnothing \qquad \text{or} \qquad C \cap \Lambda_1 \cap \overline{\Lambda_2} = \varnothing \qquad \text{or} \qquad B \cap C \cap \Lambda_1 \cap \overline{\Lambda_2} = \varnothing$$

The problem with the first and second is that they would also eliminate the possibility of *PQs* and *PRs* events respectively (see Table 6), for this reason we choose the third. This leads to similar constraints for Γ_1, Γ_2, Π_1 and Π_2 derived from *(PQa)Rs* and *(PRa)Qs* as follows:

$$A \cap C \cap \Pi_1 \cap \overline{\Pi_2} = \varnothing \qquad A \cap B \cap \overline{\Gamma_1} \cap \Gamma_2 = \varnothing$$

So summarising, if Γ_1, Γ_2, Λ_1, Λ_2, Π_1 and Π_2 satisfy the above constraints then parts of event types and complete event types are eliminated. These events must be eliminated because if one of these categories of events were present in one of the three processes then it could not in general be equivalent to either of the other two irrespective of what values were chosen for the synchronisation sets. It is also the case that adopting these constraints reduces all of the equalities on the event types which can occur to equalities of just one area in all three processes.

3.5. Associativity Law

Using the constraints and equalities derived in the previous section as well as those from the tabulation of the event types we arrive at the following associativity law for the $\|_\Omega$ operator:

$$P\|_{W\cup X\cup Y}(Q\|_{W\cup Z}R) \equiv Q\|_{W\cup X\cup Z}(P\|_{W\cup Y}R) \equiv R\|_{W\cup Y\cup Z}(P\|_{W\cup X}Q) \tag{1}$$

where $W \subseteq \Sigma$, $A \cap Z = \varnothing$, $B \cap Y = \varnothing$, $C \cap X = \varnothing$ and A, B, C are the alphabets of P, Q and R respectively. Intuitively W represents the events that all three processes are required to synchronise, X represents the events that only P and Q are required to synchronise, Y represents the events that only P and R are required to synchronise and Z represents the events that only Q and R are required to synchronise. For an outline of a proof of this law see Appendix B.

From this, the appropriate values for the original synchronisation sets are as follows:

$$\begin{array}{ll} \Lambda_1 = W \cup X \cup Y & \Lambda_2 = W \cup Z \\ \Pi_1 = W \cup X \cup Z & \Pi_2 = W \cup Y \\ \Gamma_1 = W \cup X & \Gamma_2 = W \cup Y \cup Z \end{array}$$

3.5.1. Tabulation and Alphabet Diagram of Event Types

The alphabet diagram for the associative case for the three processes is given in Figure 3. This diagram represents all three cases at once, i.e. the areas are the same for the three versions. This differs from the diagram for the general cases, Figure 2, which represented one of the three cases at a time. The event types, for events that are not inhibited in some way, are given in Table 10 and for the associative case the following event type equalities must hold:

$$\begin{array}{lllll} (PQa)Rs & = (PRa)Qs & = (QRa)Ps & = \varnothing \\ (PQs)Ra & = (PRs)Qa & = (QRs)Pa & = \varnothing \\ (PQa)i & = (PRa)i & = (QRa)i & = \varnothing \end{array}$$

Again due to space limitations we do not include the detailed tabulation of the inhibited and irrelevant events. The events which form each event type for each process can easily be derived from those in Tables 3 – 9, by substituting in the above values for the synchronisation sets.

3.6. Associativity Laws for CSP_T Parallel Operators

Based on the associativity law for the generalised parallel operator $\|_\Omega$, we can now state (proofs similar) the associativity laws for the synchronous ($\|_\Delta$), asynchronous ($|||_\Theta$) and race ($|_\Theta$) parallel operators as follows:

$$P\|_{W\cup X\cup Y}(Q\|_{W\cup Z}R) \equiv Q\|_{W\cup X\cup Z}(P\|_{W\cup Y}R) \equiv R\|_{W\cup Y\cup Z}(P\|_{W\cup X}Q) \tag{2}$$

$$P|||_{W\cup X\cup Y}(Q|||_{W\cup Z}R) \equiv Q|||_{W\cup X\cup Z}(P|||_{W\cup Y}R) \equiv R|||_{W\cup Y\cup Z}(P|||_{W\cup X}Q) \tag{3}$$

$$P|_{W\cup X\cup Y}(Q|_{W\cup Z}R) \equiv Q|_{W\cup X\cup Z}(P|_{W\cup Y}R) \equiv R|_{W\cup Y\cup Z}(P|_{W\cup X}Q) \tag{4}$$

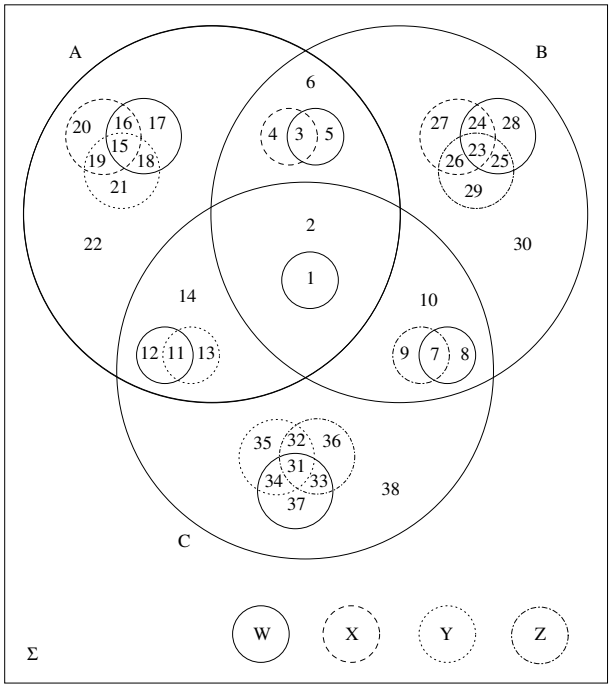

Figure 3. Alphabet diagram for associative case.

Table 10. Associative case: possible events.

Event	Set of Events	*Area*
Pa	$(A \cap \overline{B} \cap \overline{C} \cap \overline{W} \cap \overline{X} \cap \overline{Y})$	22
Qa	$(\overline{A} \cap B \cap \overline{C} \cap \overline{W} \cap \overline{X} \cap \overline{Z})$	30
Ra	$(\overline{A} \cap \overline{B} \cap C \cap \overline{W} \cap \overline{Y} \cap \overline{Z})$	38
PQa	$(A \cap B \cap \overline{C} \cap \overline{W} \cap \overline{X})$	6
PRa	$(A \cap \overline{B} \cap C \cap \overline{W} \cap \overline{Y})$	14
QRa	$(\overline{A} \cap B \cap C \cap \overline{W} \cap \overline{Z})$	10
PQRa	$(A \cap B \cap C \cap \overline{W})$	2
PQs	$(A \cap B \cap \overline{C} \cap \overline{W} \cap X)$	4
PRs	$(A \cap \overline{B} \cap C \cap \overline{W} \cap Y)$	13
QRs	$(\overline{A} \cap B \cap C \cap \overline{W} \cap Z)$	9
PQRs	$(A \cap B \cap C \cap W)$	1

In these three laws the meanings of W, X, Y, Z and A, B, C are as for $\|_\Omega$; and the same constraints apply to all three laws: $W \subseteq \Sigma$, $A \cap Z = \varnothing$, $B \cap Y = \varnothing$ and $C \cap X = \varnothing$.

However, due to the termination semantics of each operator there are additional constraints with respect to tick ($\checkmark$). For the synchronous operator we require synchronous termination, therefore, $\checkmark \in W$. In the case of the asynchronous and race operators termination is not synchronised, i.e. $\checkmark$ is an asynchronous event, therefore, $\checkmark \notin W, X, Y, Z$.

3.7. Testing for Associativity

Now that an associativity law has been proved for the generalised operator, we can use it to determine whether an instance of either $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$ or $Q\|_{\Pi_1}(P\|_{\Pi_2}R)$ or $(P\|_{\Gamma_1}Q)\|_{\Gamma_2}R$ can be transformed into an equivalent instance of the other two.

This equivalence preserving transformation can be achieved if the two synchronisation sets used to combine the three processes satisfy two conditions. The two conditions have

been derived from the way in which the pairs of synchronisation sets in the associativity law are defined in terms of the sets W, X, Y, Z and the constraints placed on each of them. This question can also be answered using the laborious technique of working out the event types which occur and checking that none of them prohibit the transformation, i.e. occur only in that particular alternative.

The two conditions for the pair of synchronisation sets for a process of the form $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$ are the following:

$$(1) \qquad A \cap \overline{\Lambda_1} \cap \Lambda_2 = \varnothing$$
$$(2)\ B \cap C \cap \Lambda_1 \cap \overline{\Lambda_2} = \varnothing$$

If Λ_1 and Λ_2 satisfy these conditions then the process can be re-written as either of the other two forms, by using Λ_1 and Λ_2 to define W, X, Y and Z. It is possible to assign different values to these sets while still ensuring equivalence, the differences only affect how events are inhibited. One set of values for these variables is the following:

$$W = \Lambda_1 \cap \Lambda_2 \qquad X = \overline{C} \cap \Lambda_1 \cap \overline{\Lambda_2} \qquad Y = \overline{B} \cap \Lambda_1 \cap \overline{\Lambda_2} \qquad Z = \overline{\Lambda_1} \cap \Lambda_2$$

These in turn are used to define the synchronisation sets for the other two processes as specified in the associativity law.

The conditions on the pairs of synchronisation sets and the definitions of W, X, Y and Z for the (Π) and (Γ) versions are as follows:

$$(1) \qquad B \cap \overline{\Pi_1} \cap \Pi_2 = \varnothing$$
$$(2)\ A \cap C \cap \Pi_1 \cap \overline{\Pi_2} = \varnothing$$

$$W = \Pi_1 \cap \Pi_2 \qquad X = \overline{C} \cap \Pi_1 \cap \overline{\Pi_2} \qquad Y = \overline{\Pi_1} \cap \Pi_2 \qquad Z = \overline{A} \cap \Pi_1 \cap \overline{\Pi_2}$$

$$(1) \qquad C \cap \Gamma_1 \cap \overline{\Gamma_2} = \varnothing$$
$$(2)\ A \cap B \cap \overline{\Gamma_1} \cap \Gamma_2 = \varnothing$$

$$W = \Gamma_1 \cap \Gamma_2 \qquad X = \Gamma_1 \cap \overline{\Gamma_2} \qquad Y = \overline{B} \cap \overline{\Gamma_1} \cap \Gamma_2 \qquad Z = \overline{A} \cap \overline{\Gamma_1} \cap \Gamma_2$$

So for example, assuming Λ_1 and Λ_2 satisfy both conditions then the following holds:

$$P\|_{\Lambda_1}(Q\|_{\Lambda_2}R) \ \equiv\ Q\|_{\Pi_1}(P\|_{\Pi_2}R) \ \equiv\ (P\|_{\Gamma_1}Q)\|_{\Gamma_2}R$$

And the definition of the other synchronisation sets are as follows:

$$\Pi_1 = (\Lambda_1 \cap \Lambda_2) \cup (\overline{C} \cap \Lambda_1 \cap \overline{\Lambda_2}) \cup (\overline{\Lambda_1} \cap \Lambda_2) = \Lambda_2 \cup (\Lambda_1 \cap \overline{C})$$
$$\Pi_2 = (\Lambda_1 \cap \Lambda_2) \cup (\overline{B} \cap \Lambda_1 \cap \overline{\Lambda_2}) = (\Lambda_1 \cap \Lambda_2) \cup (\Lambda_1 \cap \overline{B})$$
$$\Gamma_1 = (\Lambda_1 \cap \Lambda_2) \cup (\overline{C} \cap \Lambda_1 \cap \overline{\Lambda_2}) = (\Lambda_1 \cap \Lambda_2) \cup (\Lambda_1 \cap \overline{C})$$
$$\Gamma_2 = (\Lambda_1 \cap \Lambda_2) \cup (\overline{B} \cap \Lambda_1 \cap \overline{\Lambda_2}) \cup (\overline{\Lambda_1} \cap \Lambda_2) = \Lambda_2 \cup (\Lambda_1 \cap \overline{B})$$

4. Specifying and Implementing the Parallel Composition of Three Processes

In this section we show how to specify a parallel composition of three processes in terms of event types, and implement it using $\|_\Omega$ by defining the two synchronisation sets required.

The specification of the parallel composition of processes in terms of event types can be viewed as the specification of the inter-connections/interface between a network of three processes. Here we only consider a network of three processes, but we believe the present

treatment could be extended to deal with a network consisting of an arbitrary number of processes.

This reverses the type of analysis performed during the search for the associativity law, where given the processes and two synchronisation sets, it was defined what type each event would be. The present approach is more useful when using $\|_\Omega$ to implement networks of processes, whereas the other is more useful when analysing existing ones.

The way in which we specify the parallel composition and interface of three processes in terms of event types is done in three stages:

1. Define the alphabets A, B and C of the three processes P, Q and R.
2. Determine how the events in each of the following seven regions $A \cap \overline{B} \cap \overline{C}$, $\overline{A} \cap B \cap \overline{C}$, $\overline{A} \cap \overline{B} \cap C$, $A \cap B \cap \overline{C}$, $A \cap \overline{B} \cap C$, $\overline{A} \cap B \cap C$ and $A \cap B \cap C$ should be partitioned into event types
3. Define the two synchronisation sets using these partitions.

We shall now give an example of how to apply these three stages for the associative case.

4.1. Example for the Associative Case

Our aim is to illustrate how to choose two interfaces (synchronisation sets) I_1 and I_2, such that we can apply the associativity law to the following simple network:

$$P\|_{I_1}(Q\|_{I_2}R)$$

Thus, we need to ensure that the event types that cause the associativity law to fail, for this form of network, cannot arise with our chosen interfaces.

Stage one is to define the alphabets for the three processes. For this example, we chose the arbitrary alphabets A, B and C respectively. Hence, based on our previous investigation, we must ensure that certain categories of *Pa* events and both *(QRa)Ps* and *(QRs)Pa* events do not occur. Therefore, we need to define the sets W, X, Y and Z as used in law (1) to define the two interfaces as follows:

$$I_1 = W \cup X \cup Y \qquad \text{and} \qquad I_2 = W \cup Z$$

where the W, X, Y and Z have the same intuitive meanings and satisfy the same constraints as given previously. By defining these four sets in this way, we are ensuring that the associativity law is applicable for our two interfaces.

For stage two, we use Table 11 to tabulate the events and components of events which fall into the seven areas. Note that we have ignored *irrelevant* synchronisation events and *impossible* events. As an example of how to read the table consider the region $A \cap \overline{B} \cap \overline{C}$; if the (Λ) form of process has been chosen the events in this area will be either *Pa* or *Pi* events, depending on how it is partitioned. A diagrammatic view of how the seven areas are partitioned for the (Λ) process is given in Figure 4.

There are a number of equalities which hold between the event types in each alternative and between the three alternatives. These additional equalities hold because we are dealing with the associative case. For simplicity we do not distinguishing between the different ways in which an event is inhibited, since for most cases this is irrelevant. Therefore, we group together the various categories of inhibited events using the following six equalities denoted by *Pw*, *Qw*, *Rw*, *PQw*, *PRw* and *QRw*. These inhibited events are then included in the synchronisation set W.

Table 11. Associate case: partitioned events

Area	(Λ)	(Π)	(Γ)
$A \cap \overline{B} \cap \overline{C}$	Pa, Pi_1	$Pa, Pi_1, (Pa)i$	$Pa, Pi_1, (Pa)i$
$\overline{A} \cap B \cap \overline{C}$	$Qa, Qi_1, (Qa)i$	Qa, Qi_1	$Qa, Qi_1, (Qa)i$
$\overline{A} \cap \overline{B} \cap C$	$Ra, Ri_1, (Ra)i$	$Ra, Ri_1, (Ra)i$	Ra, Ri_1
$A \cap B \cap \overline{C}$	PQa, PQs, Pi_2, Qi_2	PQa, PQs, Pi_2, Qi_2	$PQa, PQs, (PQs)i$
$A \cap \overline{B} \cap C$	PRa, PRs, Pi_3, Ri_2	$PRa, PRs, (PRs)i$	PRa, PRs, Pi_2, Ri_2
$\overline{A} \cap B \cap C$	$QRa, QRs, (QRs)i$	QRa, QRs, Qi_3, Ri_2	QRa, QRs, Qi_2, Ri_3
$A \cap B \cap C$	$PQRa, PQRs$	$PQRa, PQRs$	$PQRa, PQRs$

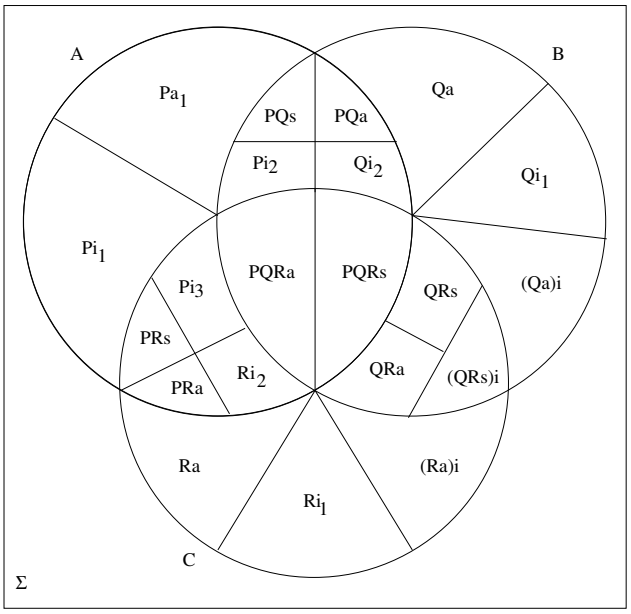

Figure 4. Partition of events for associative case of $P\|_{\Lambda_1}(Q\|_{\Lambda_2}R)$.

$$
\begin{array}{lrcccl}
A \cap \overline{B} \cap \overline{C}: & Pw & = & Pi_1^{\Lambda} & = Pi_1^{\Pi} \cup (Pa)i^{\Pi} & = Pi_1^{\Gamma} \cup (Pa)i^{\Gamma} \\
\overline{A} \cap B \cap \overline{C}: & Qw & = & Qi_1^{\Lambda} \cup (Qa)i^{\Lambda} & = Qi_1^{\Pi} & = Qi_1^{\Gamma} \cup (Qa)i^{\Gamma} \\
\overline{A} \cap \overline{B} \cap C: & Rw & = & Ri_1^{\Lambda} \cup (Ra)i^{\Lambda} & = Ri_1^{\Pi} \cup (Ra)i^{\Pi} & = Ri_1^{\Gamma} \\
A \cap B \cap \overline{C}: & PQw & = & Pi_2^{\Lambda} = Qi_2^{\Lambda} & = Pi_2^{\Pi} = Qi_2^{\Pi} & = (PQs)i^{\Gamma} \\
A \cap \overline{B} \cap C: & PRw & = & Pi_3^{\Lambda} = Ri_2^{\Lambda} & = (PRs)i^{\Pi} & = Pi_2^{\Gamma} = Ri_2^{\Gamma} \\
\overline{A} \cap B \cap C: & QRw & = & (QRs)i^{\Lambda} & = Qi_3^{\Pi} = Ri_2^{\Pi} & = Qi_2^{\Gamma} = Ri_3^{\Gamma}
\end{array}
$$

Finally for stage three, we define the sets W, X, Y and Z to be used to define the interfaces I_1 and I_2 as follows:

$$W = PQRs \cup (Pw \cup Qw \cup Rw \cup PQw \cup PRw \cup QRw)$$

$$X = PQs \qquad Y = PRs \qquad Z = QRs$$

There are obviously other ways of combining the synchronous and inhibited events to achieve the desired result but we leave the definition of these to the reader.

5. Further Work

We have demonstrated how to use $\|_{\Omega}$ to implement a network of three processes with specific interfaces between the processes. These interfaces were defined in terms of the event types

of the processes' alphabets. This approach entailed defining the processes' alphabets, then partitioning up the alphabets into the required event types and finally defining the two synchronisation sets using these event types. An obvious extension of this work is to develop a method, and probably more importantly some succinct notation, that would allow an arbitrary number of processes to be constructed into a network with the required event types present. For example, with n processes the network would have the form:

$$P_1 \parallel_{\Omega_1} (P_2 \parallel_{\Omega_2} (\ldots (P_{n-1} \parallel_{\Omega_{n-1}} P_n) \ldots)$$

The notation would need to be able to deal with the n alphabets of the processes, the $n-1$ synchronisation sets and the 2^{2n-1} different event types that could occur, although most of these event types would be classed as *inhibited*.

The process of constructing the required network would then be similar to that for three processes given in Section 4, and in outline would be as follows:

1. Define the alphabets $A_1, \ldots, A_n$ of the n processes, $P_1, \ldots, P_n$.
2. Determine how the events in each of the 2^n regions $A_1 \cap \overline{A}_2 \cap \ldots \cap \overline{A}_n$ to $\overline{A}_1 \cap \ldots \cap \overline{A}_{n-1} \cap A_n$ should be partitioned into event types.
3. Finally, define the $n-1$ synchronisation sets $\Omega_1, \ldots, \Omega_{n-1}$ using these partitions.

If the network was required to be "associative", in the sense that the order in which the processes were to be composed was irrelevant, then its construction would be greatly simplified. Since, the sets of synchronisation events between each pair of processes is one of the most important factors in ensuring associativity, we could use $X_{i,j}$ to denote the required set of synchronous events between P_i and P_j. Then for each $X_{i,j}$ require that it is disjoint with all other processes' alphabets in the network, i.e. the following condition should hold:

$$X_{i,j} \cap (\bigcup_{k \neq i,j} A_k) \;=\; \varnothing$$

We are grateful to one of the anonymous reviewers who raised the issue of whether it is possible to give an indication of the "order of magnitude" of the different types of events present. Currently, we are only able to provide an answer for the most important event types with respect to ensuring the associativity law holds, namely the *pure* synchronous and asynchronous events. In this context, pure means those types that are performed either synchronously by all participating processes, e.g. *PQRs*, or asynchronously by all processes, e.g. *PQRa*. For these we are able to state that: for n processes the total number of different (pure) synchronous event types is $2^n - (n+1)$ and for (pure) asynchronous event types it is $2^n - 1$. We leave the derivation of similar results for the more complex case for *mixed* event types, i.e. those that are performed synchronously by some processes and asynchronously by others, e.g. *(PQa)Rs*, as a topic for future research.

We are grateful to two of the anonymous reviewers who raised issues regarding the constraints on the two synchronisation sets that are required to ensure that the associativity law holds. In particular, are the conditions we give in Section 3.4 necessary? They are clearly sufficient, but currently we are unable to say with any confidence that they are also necessary. However, given that we consider a number of alternative conditions, and that one reviewer has also suggested:

$$\bar{C} \cap \Lambda_2 = \bar{B} \cap \Lambda_2 = \varnothing \qquad\qquad (\text{equivalently} \quad \Lambda_2 \;\subseteq\; B \cap C),$$

as an alternative constraint to eliminate the problem *Pa* events; we strongly suspect that they are not.

6. Summary and Conclusions

We have extended previous work describing our language and model CSP_T for specifying the behaviour of networks of processes which is a variant (a variant specifically chosen to be as close as possible to the original formulation) of the original language of CSP developed by Hoare, Roscoe and others. Specifically, we have extended our analysis by the discovery and proof of associativity laws for the three parallel operators. By developing these laws we provide specifiers and designers with essential laws for specifying, designing, analysing and reasoning about networks of parallel processes.

In particular, our analysis of the event types which can occur in processes of the form $(P\|_{\Gamma_1}Q)\|_{\Gamma_2}R$, resulted in the discovery of constraints on the pair of synchronisation sets Γ_1 and Γ_2, which when satisfied allowed the definition of an associativity law. These constraints also provided us with a simple "test" for associativity. As a result of this analysis we were able to prove the general associativity law for $\|_\Omega$ and hence, for $\|_\Delta$, $|||_\Theta$ and $|_\Theta$. We believe these laws, although not "universally applicable" in the sense of Roscoe [5], that it is the most general and useful possible for these style of operators.

Furthermore, we demonstrated how to use $\|_\Omega$ to implement a network of three processes with specific interfaces (synchronisation sets) between the processes. These interfaces are defined in terms of the event types of the processes' alphabets. This approach entails defining the processes' alphabets, then partitioning up the alphabets into the required event types and finally defining the two interfaces using these event types. Clearly, this would be a useful technique when attempting to construct networks of processes using $\|_\Omega$. We believe this approach could be scaled up to deal with an arbitrary number of processes, with the aid of some well chosen notation, but we leave the details as a topic for further investigation.

Based on our investigations, we suggest the following general guideline for system designers, when choosing interfaces for these type of networks: if one wishes to be able to apply an associativity law, then one should avoid any interface that results in *mixed* event types, as these can only occur in networks that are constructed in particular ways.

Note that associativity is not always desired and that, in those cases, events can be (and need to be) of mixed type. For example, client-server systems have many client processes interleaving access to a pair of request and reply channels: first they send a request, then wait for a reply. Their use of those channels is asynchronous between them. The set of client processes (i.e. their parallel composition) now synchronises with the (single) server process on those request and reply channels: the server waits for a request, then sends a reply. The reply will be taken by the client that made the request since it will be the only client listening - the other clients are either blocked trying to make a request or doing something else. So, the request and reply channel interfaces are mixed: asynchronous between the clients and synchronous between the client group and the server. Associating the component processes in some other way (e.g. with some clients grouped with the server and some just with themselves) gives a system that is semantically *different* and incorrect[5].

However, when associativity matters, by identifying the associativity constraints and proving the set of associativity laws, we have provided a practical technique for designers to capture intuitively and flexibly the intended behaviour of the processes networks they wish to build.

Acknowledgements

We would like to acknowledge and thank the three anonymous referees for their feedback on the paper. We found this very helpful in improving the paper overall, but especially in

[5]This example was suggested by one of the anonymous reviewers.

improving the readability of the paper. We would also like to gratefully acknowledge the advice and assistance of the following individuals from whom at various stages we benefited a great deal over several very detailed discussions on our work: Richard Bornat, Jonathan Bowen, Michael Luck and Steve Schneider. Finally, we would like to acknowledge the help and assistance of the CPA 2013 Editors, especially Peter Welch, in preparing this paper.

References

[1] S. D. Brookes. *A Model for Communicating Sequential Processes*. PhD thesis, Oxford University, 1983.
[2] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(7), 1985.
[3] S. D. Brookes and A. W. Roscoe. An improved failures model for Communicating Sequential Processes. In *Proceedings of Pittsburgh Seminar on Concurrency*, LNCS 197, pages 281–305. Springer-Verlag, 1985.
[4] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985. ISBN: 0-131-53271-5.
[5] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1997. ISBN: 0-13-674409-5.
[6] A.W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010. ISBN: 978-1-84882-257-3.
[7] P. Howells and M. d'Inverno. A CSP model with flexible parallel termination semantics. *Formal Aspects of Computing*, 21(5):421–449, 2009.
[8] P. Howells and M. d'Inverno. Specifying Termination in CSP. *Theoretical Computer Science*, 2013. DOI: 10.1016/j.tcs.2013.05.008.
[9] P. Howells and Mark d'Inverno. Successful Termination in Timed CSP. In *Proceedings of Communicating Process Architectures 2013 (CPA13)*, 2013.
[10] P. Howells. *Communicating Sequential Processes with Flexible Parallel Termination Semantics*. PhD thesis, University of Westminster, 2005.
[11] H. Tej and B. Wolff. A Corrected Failure-Divergence Model for CSP in Isabelle/HOL. In *Proceedings of the FME '97 – Industrial Applications and Strengthened Foundations of Formal Methods*, LNCS 1313. Springer-Verlag, 1997.
[12] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.
[13] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 2000. ISBN: 0-471-62373-3.
[14] J. W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice-Hall International, 1980.
[15] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

A. (Appendix) CSP$_T$: Notation, Axioms and Semantic Definitions

This appendix contains a summary of the notation and definitions used in the paper and the proof of the associativity law for the generalised parallel operator $\|_\Omega$. For a more detailed description of CSP, see [4,5,6] and for CSP$_T$ see [7,8].

A.1. Basic Notation

The notation used in CSP and CSP$_T$ is the following. Σ is the set of all events, and is countable; denoted by a, b, c. $\mathbb{P}(\Sigma)$ is the power set of Σ; denoted by X, Y, Z. Σ^* is the set of finite sequences of events, i.e., *traces*; denoted by r, s, t, u. $\langle\,\rangle$ represents the empty sequence. $\langle a, b, c, d\rangle$ represents the sequence with members a, b, c, d. $s^\frown t$ represents the concatenation of the two sequences s and t. $s \leq t$ is the sequence *prefix* relation and $s < t$ is the proper *prefix* relation. $\#s$ is the length of the sequence s. $a \text{ in } s$ is sequence membership. $\overline{X}$ represents the set complement of X, with respect to Σ, i.e., $\overline{X} \cap X = \varnothing$ and $\overline{X} \cup X = \Sigma$. $X - Y$ is the set difference of X and Y, i.e., $X - Y = X \cap \overline{Y}$. Finally, processes are denoted by P, Q, R and A, B, C denote their alphabets. The set of all process identifiers is denoted by Ide, and use $p, q, \ldots$ to range over Ide.

A.2. Axioms for Failures and Divergences

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:

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

In addition, they must also satisfy the following *process axioms*, including the CSP$_T$ *Termination* axiom (T1):

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

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

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

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

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

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

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

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

For a natural language interpretation of these axioms, see the Appendix of [9].

Definition A.1 The traces and alphabet of a process are now defined as follows:

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

A.3. Nondeterministic (refinement) ordering on processes

The *refinement ordering* on CSP processes is defined in terms of the reverse subset ordering on the failure and divergence set components of processes, (the details of the underlying theory can be found in [14,15]), and is defined as follows.

Definition A.2 The refinement ordering $\sqsubseteq$ is defined as follows:

$$\langle F_1, D_1\rangle \sqsubseteq \langle F_2, D_2\rangle \;\;\widehat{=}\;\; F_1 \supseteq F_2 \wedge D_1 \supseteq D_2$$

where $\langle F_1, D_1\rangle$ and $\langle F_2, D_2\rangle$ represent the processes P_1 and P_2 respectively.

$P_1 \sqsubseteq P_2$, (P_1 *is refined by* P_2) means that P_1 is *more nondeterministic* than P_2 if it can diverge whenever P_2 can diverge and fail whenever P_2 can fail, i.e. any behaviour which is possible for P_2 is also possible for P_1 and that P_1 may also exhibit behaviour which is not possible for P_2.

Definition A.3 The ordering $\sqsubseteq$ induces an equivalence relation $\equiv$ on processes defined as follows:

$$P \;\equiv\; Q \;\;\widehat{=}\;\; P \sqsubseteq Q \wedge Q \sqsubseteq P$$

Hence two processes are equivalent in the model N_T when their failure and divergent set components are equal. This space of processes together with the ordering, i.e., ($PROC_T$, $\sqsubseteq$) is a *partially ordered set*, with least element $\perp$.

A.4. Semantic functions for CSP$_T$

This section contains the semantics functions $\mathcal{F}$ and $\mathcal{D}$ for the generalised parallel operator $P\|_\Omega Q$. The CSP$_T$ semantic functions for the other "standard" processes and operators, the two new parallel operators: $P|||_\Theta Q$ and $P|_\Theta Q$, can be found in [7] and for CSP in [4,5,6].

First, we define an environment that is used to define process identifiers and the failure and divergence semantics functions as follows.

Definition A.4 An *environment e* is a function which maps process identifiers (Ide) to failure and divergence pairs $\langle F, D\rangle$ as follows.

$$e \ : \ \text{Ide} \to \mathbb{P}(\Sigma^* \times \mathbb{P}(\Sigma)) \times \mathbb{P}(\Sigma^*)$$

The set of all such environments is denoted by NEnv.

Definition A.5 The semantic function $\mathcal{N}$ for CSP$_T$ is defined as follows:

$$\mathcal{N} \ : \ \mathbf{CSP}_T \to [\text{NEnv} \to N_T]$$
$$\mathcal{N}[\![P]\!]e \ = \ \langle \mathcal{F}[\![P]\!]e, \ \mathcal{D}[\![P]\!]e\rangle$$

where the two functions $\mathcal{F}$ and $\mathcal{D}$ are defined as follows:

$$\mathcal{F} \ : \ \mathbf{CSP}_T \to [\text{NEnv} \to \mathbb{P}(\Sigma^* \times \mathbb{P}(\Sigma))]$$
$$\mathcal{D} \ : \ \mathbf{CSP}_T \to [\text{NEnv} \to \mathbb{P}(\Sigma^*)]$$

We now give the definition of the semantic functions $\mathcal{F}$ and $\mathcal{D}$ for $\|_\Omega$ (and hence $\|_\Delta$). The semantic functions for $P\|_\Omega Q$ use the merge function to construct the *legal* traces for the parallel operator. merge takes three arguments, two traces and a synchronisation set, and maps them to the (possibly empty) set of traces. If the result of merging two traces, s and t, is the empty set, then the two traces are *incompatible*, because of synchronous events.

Definition A.6 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 $a \neq b$.

$$\begin{aligned}
\mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) &= \mathsf{merge}(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Omega) && [a, b \notin \Omega]\\
&= \{\langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, \langle b\rangle^\frown t, \Omega)\}\\
&\quad \cup \{\langle b\rangle^\frown u \mid u \in \mathsf{merge}(\langle a\rangle^\frown s, t, \Omega)\}\\
\mathsf{merge}(\langle a\rangle^\frown s, \langle a\rangle^\frown t, \Omega) &= \{\ \langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, t, \Omega)\ \} && [a \in \Omega]\\
\mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) &= \mathsf{merge}(\langle b\rangle^\frown t, \langle a\rangle^\frown s, \Omega) && [a \notin \Omega, b \in \Omega]\\
&= \{\ \langle a\rangle^\frown u \mid u \in \mathsf{merge}(s, \langle b\rangle^\frown t, \Omega)\ \}\\
\mathsf{merge}(\langle a\rangle^\frown s, \langle b\rangle^\frown t, \Omega) &= \varnothing && [a, b \in \Omega]\\
\mathsf{merge}(\langle\,\rangle, \langle a\rangle^\frown s, \Omega) &= \mathsf{merge}(\langle a\rangle^\frown s, \langle\,\rangle, \Omega) && [a \notin \Omega]\\
&= \{\ \langle a\rangle^\frown u \mid u \in \mathsf{merge}(\langle\,\rangle, s, \Omega)\ \}\\
\mathsf{merge}(\langle\,\rangle, \langle a\rangle^\frown s, \Omega) &= \mathsf{merge}(\langle a\rangle^\frown s, \langle\,\rangle, \Omega) \ = \ \varnothing && [a \in \Omega]\\
\mathsf{merge}(\langle\,\rangle, \langle\,\rangle, \Omega) &= \{\ \langle\,\rangle\ \}
\end{aligned}$$

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

$$\mathcal{D}[\![P\|_\Omega Q]\!]e = \{\, u^\frown v \mid \exists\, s,t : u \in \mathsf{merge}(s,t,\Omega) \wedge ((s \in \mathcal{D}[\![P]\!]e \wedge t \in \mathsf{traces}(Q)) \vee (s \in \mathsf{traces}(P) \wedge t \in \mathcal{D}[\![Q]\!]e)) \,\}$$

$$\mathcal{F}[\![P\|_\Omega Q]\!]e = \{\, (u, X\cup Y\cup Z) \mid X \subseteq \overline{\Omega} \wedge Y,Z \subseteq \Omega \wedge \exists\, s,t : u \in \mathsf{merge}(s,t,\Omega) \wedge (s, X\cup Y) \in \mathcal{F}[\![P]\!]e \wedge (t, X\cup Z) \in \mathcal{F}[\![Q]\!]e \,\} \cup \{\, (u,X) \mid u \in \mathcal{D}[\![P\|_\Omega Q]\!]e \wedge X \in \mathbb{P}(\Sigma) \,\}$$

B. (Appendix) Proof of the Associativity Law

In this section we present an outline of just the following part of the proof of:

$$P\|_{W\cup X\cup Y}(Q\|_{W\cup Z}R) \;\equiv\; R\|_{W\cup Y\cup Z}(P\|_{W\cup X}Q) \tag{1}$$

The proof of equivalence of the other process with either of these two is similar. Equation (1) is proved[6] by showing that the divergence and failure sets of the two sides are equal. First, we need some lemmas.

B.1. Merge Lemmas

To prove the associativity law we require the following merge lemmas: symmetry, trace prefix and associativity.

$$\mathsf{merge}(s,t,\Omega) = \mathsf{merge}(t,s,\Omega) \tag{merge-Sym}$$

$$s^\frown t \in \mathsf{merge}(u,w,\Omega) \Rightarrow \exists\, u', w' : u' \leq u \wedge w' \leq w \wedge s \in \mathsf{merge}(u', w', \Omega) \tag{merge-Prefix}$$

[6]One of the anonymous reviewers suggested that our law (1) was a special case of Roscoe's $\langle \|_A \|_B - \mathrm{assoc}\rangle$ law, see Section 3.2, and suggested the following proof:

Let D, E, F be the private asynchronous events of P, Q, R. Then the left-hand side equals:

$$P\ {}_{D\cup W\cup X\cup Y}\|_{E\cup F\cup W\cup X\cup Y\cup Z}\ (\,Q\ {}_{E\cup W\cup X\cup Z}\|_{F\cup W\cup Y\cup Z}\ R\,)$$

using the law relating alphabetised and generalised parallel (see Section 3.2). Then by Law $\langle \|_A \|_B - \mathrm{assoc}\rangle$, the above equals:

$$(\,P\ {}_{D\cup W\cup X\cup Y}\|_{E\cup W\cup X\cup Z}\ Q\,)\ {}_{D\cup E\cup W\cup X\cup Y\cup Z}\|_{F\cup W\cup Y\cup Z}\ R$$

Which equals the right-hand side, again by the law relating alphabetised and generalised parallel. In particular, the above shows that the law you produce is just a particular instance of the law for alphabetised parallel.

We agree that this equality above is correct. However, we believe that the sets used do not include any of the shared asynchronous events between the three processes, i.e. *PQa*, *PRa*, *QRa* or *PQRa*. Furthermore, if these were included in the above example, then due to the definition of this operator they would automatically become synchronised events, since they are all within the intersections of the sets used. In particular, (assuming that the W, X, Y are as defined in our law) then $D \cup W \cup X \cup Y$ does not necessarily define the alphabet of P. Since even with the inclusion of P's private asynchronous events D, this set does not include P's shared asynchronous events with either Q (*PQa*) or R (*PRa*) or both (*PQRa*). Therefore, we believe that Roscoe's law $\langle \|_X - \mathrm{assoc}\rangle$ is a special case of our associativity law, since ours also deals with this more general case when shared asynchronous events are present, which Roscoe's law does not.

$$\begin{aligned}
&\{\, u \mid u \in \mathsf{merge}(p, x, W \cup X \cup Y) \wedge x \in \mathsf{merge}(q, r, W \cup Z) \,\} && \text{(merge-Ass)}\\
=&\{\, u \mid u \in \mathsf{merge}(q, y, W \cup X \cup Z) \wedge y \in \mathsf{merge}(p, r, W \cup Y) \,\}\\
=&\{\, u \mid u \in \mathsf{merge}(r, z, W \cup Y \cup Z) \wedge z \in \mathsf{merge}(p, q, W \cup X) \,\}
\end{aligned}$$

where W, X, Y, Z, A, B and C are as before and $p \in A^*$, $q \in B^*$ and $r \in C^*$.

We omit the poofs of these lemmas as they are straightforward (by structural induction) and lengthy. The details can be found in [10]. □

B.2. Proof of Divergence Set Equality

We need to prove the following:

$$\mathcal{D}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \;=\; \mathcal{D}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e$$

This proof is split into the following two parts:

(a) $s \in \mathcal{D}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \;\Rightarrow\; s \in \mathcal{D}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e$
(b) $s \in \mathcal{D}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e \;\Rightarrow\; s \in \mathcal{D}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e$

To prove (a) we start by expanding the *lhs* using the definition of $\mathcal{D}[\![P \|_\Omega Q]\!]e$.

$$\begin{aligned}
&\mathcal{D}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e\\
=\;& \{u^\frown v \mid \exists p, q, r, x : u \in \mathsf{merge}(p, x, W \cup X \cup Y) \wedge p \in \mathcal{D}[\![P]\!]e\\
&\qquad \wedge x \in \mathsf{merge}(q, r, W \cup Z) \wedge q \in \mathsf{traces}(Q) \wedge r \in \mathsf{traces}(R)\}\\
&\cup \{u^\frown v \mid \exists p, q, r, x, y : u \in \mathsf{merge}(p, x^\frown y, W \cup X \cup Y) \wedge p \in \mathsf{traces}(P)\\
&\qquad \wedge x \in \mathsf{merge}(q, r, W \cup Z) \wedge q \in \mathcal{D}[\![Q]\!]e \wedge r \in \mathsf{traces}(R)\}\\
&\cup \{u^\frown v \mid \exists p, q, r, x, y : u \in \mathsf{merge}(p, x^\frown y, W \cup X \cup Y) \wedge p \in \mathsf{traces}(P)\\
&\qquad \wedge x \in \mathsf{merge}(q, r, W \cup Z) \wedge q \in \mathsf{traces}(Q) \wedge r \in \mathcal{D}[\![R]\!]e\}
\end{aligned}$$

We shall refer to these three sets as (*lhs-P*), (*lhs-Q*) and (*lhs-R*) respectively. The *rhs* can be expanded similarly as follows:

$$\begin{aligned}
&\mathcal{D}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e\\
=\;& \{u^\frown v \mid \exists p, q, r, z : u \in \mathsf{merge}(r, z, W \cup Y \cup Z) \wedge r \in \mathcal{D}[\![R]\!]e\\
&\qquad \wedge z \in \mathsf{merge}(p, q, W \cup X) \wedge p \in \mathsf{traces}(P) \wedge q \in \mathsf{traces}(Q)\}\\
&\cup \{u^\frown v \mid \exists p, q, r, z, y : u \in \mathsf{merge}(r, z^\frown y, W \cup Y \cup Z) \wedge r \in \mathsf{traces}(R)\\
&\qquad \wedge z \in \mathsf{merge}(p, q, W \cup X) \wedge p \in \mathcal{D}[\![P]\!]e \wedge q \in \mathsf{traces}(Q)\}\\
&\cup \{u^\frown v \mid \exists p, q, r, z, y : u \in \mathsf{merge}(r, z^\frown y, W \cup Y \cup Z) \wedge r \in \mathsf{traces}(R)\\
&\qquad \wedge z \in \mathsf{merge}(p, q, W \cup X) \wedge p \in \mathsf{traces}(P) \wedge q \in \mathcal{D}[\![Q]\!]e\}
\end{aligned}$$

We shall refer to these three sets as (*rhs-R*), (*rhs-P*) and (*rhs-Q*) respectively. Note that both the *rhs* and the *lhs* divergent sets are the union of three sets and that each one represents the case where one of the processes is definitely diverging. To prove part (a) it is sufficient to show that each of the *rhs* sets is a subset of the corresponding *lhs* sets, so we must prove the following:

(i) $s \in (lhs\text{-}P) \;\Rightarrow\; s \in (rhs\text{-}P)$
(ii) $s \in (lhs\text{-}Q) \;\Rightarrow\; s \in (rhs\text{-}Q)$
(iii) $s \in (lhs\text{-}R) \;\Rightarrow\; s \in (rhs\text{-}R)$

The proof of (i) is as follows:

(1)	$s \in (lhs\text{-}P)$	Assumption
(2)	$\exists\, u,p,q,r,x : u \leq s \wedge u \in \mathsf{merge}(p,x,W \cup X \cup Y)$ $\wedge\, x \in \mathsf{merge}(q,r,W \cup Z) \wedge p \in \mathcal{D}[\![P]\!]e$ $\wedge\, q \in \mathsf{traces}(Q) \wedge r \in \mathsf{traces}(R)$	
(3)	$\exists\, u,p,q,r,z : u \leq s \wedge u \in \mathsf{merge}(r,z,W \cup Y \cup Z)$ $\wedge\, z \in \mathsf{merge}(p,q,W \cup X) \wedge p \in \mathcal{D}[\![P]\!]e$ $\wedge\, q \in \mathsf{traces}(Q) \wedge r \in \mathsf{traces}(R)$	(merge-Sym), (merge-Ass)
(4)	$s \in (rhs\text{-}P)$	$y = \langle\,\rangle$, $(rhs\text{-}P)$

This completes the proof of part (i). □

The proof of (ii) is as follows:

(1)	$s \in (lhs\text{-}Q)$	Assumption
(2)	$\exists\, u,p,q,r,x,y : u \leq s \wedge u \in \mathsf{merge}(p, x^\frown y, W \cup X \cup Y)$ $\wedge\, x \in \mathsf{merge}(q,r,W \cup Z) \wedge p \in \mathsf{traces}(P)$ $\wedge\, q \in \mathcal{D}[\![Q]\!]e \wedge r \in \mathsf{traces}(R)$	
(3)	$u \in \mathsf{merge}(p, x^\frown y, W \cup X \cup Y) \wedge p \in \mathsf{traces}(P)$	
(4)	$\exists\, u',p' : u' \leq u \wedge p' \leq p \wedge u' \in \mathsf{merge}(p',x,W \cup X \cup Y)$ $\wedge\, p' \in \mathsf{traces}(P)$	(merge-Prefix), N2(P)
(5)	$\exists\, u',p',q,r,x : u' \leq s \wedge u' \in \mathsf{merge}(p',x,W \cup X \cup Y)$ $\wedge\, x \in \mathsf{merge}(q,r,W \cup Z) \wedge p' \in \mathsf{traces}(P)$ $\wedge\, q \in \mathcal{D}[\![Q]\!]e \wedge r \in \mathsf{traces}(R)$	(2), (4)
(6)	$\exists\, u',p',q,r,z : u' \leq s \wedge u' \in \mathsf{merge}(r,z,W \cup Y \cup Z)$ $\wedge\, z \in \mathsf{merge}(p',q,W \cup X) \wedge r \in \mathsf{traces}(R)$ $\wedge\, p' \in \mathsf{traces}(P) \wedge q \in \mathcal{D}[\![Q]\!]e$	(merge-Ass)
(7)	$s \in (rhs\text{-}Q)$	$y = \langle\,\rangle$, $(rhs\text{-}Q)$

This completes the proof of part (ii). □

The proof of (iii) is similar and is omitted. Therefore, this completes the proof of part (a).

The proof of part (b) is similar and is omitted. Thus we have proved that the divergent sets of both sides are equal as required.

B.3. Proof of Failure Set Equality

We need to prove the following:

$$\mathcal{F}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \;=\; \mathcal{F}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e$$

Again the proof is split into the following two parts:

(a) $(s,X) \in \mathcal{F}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \;\Rightarrow\; (s,X) \in \mathcal{F}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e$;
(b) $(s,X) \in \mathcal{F}[\![R \|_{W\cup Y\cup Z} (P \|_{W\cup X} Q)]\!]e \;\Rightarrow\; (s,X) \in \mathcal{F}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e$.

Only the proof of (a) is given since that of (b) is similar. To prove (a) we start by expanding the *lhs* repeatedly using the definition of $\mathcal{F}[\![P \|_\Omega Q]\!]e$:

$$\begin{aligned}
&\mathcal{F}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \\
=\;& \{\, (u, D \cup E \cup F) \mid \exists\, p,x : D \subseteq \overline{W} \cap \overline{X} \cap \overline{Y} \wedge E,F \subseteq W \cup X \cup Y \\
&\qquad \wedge\, u \in \mathsf{merge}(p,x,W \cup X \cup Y) \wedge (p, D \cup E) \in \mathcal{F}[\![P]\!]e \\
&\qquad \wedge\, (x, D \cup F) \in \mathcal{F}[\![Q \|_{W\cup Z} R]\!]e \,\} \\
&\cup \{\, (u,D) \mid u \in \mathcal{D}[\![P \|_{W\cup X\cup Y} (Q \|_{W\cup Z} R)]\!]e \,\}
\end{aligned}$$

Since the divergence sets were dealt with above we can ignore the second set.

$$\begin{aligned}
= \quad & \{\, (u, D \cup E \cup F) \mid \exists p, q, r, x : D \subseteq \overline{W} \cap \overline{X} \cap \overline{Y} \wedge E, F \subseteq W \cup X \cup Y \\
& \qquad \wedge D \cup F = U \cup V \cup M \wedge U \subseteq \overline{W} \cap \overline{Z} \\
& \qquad \wedge V, M \subseteq W \cup Z \wedge u \in \mathsf{merge}(p, x, W \cup X \cup Y) \\
& \qquad \wedge x \in \mathsf{merge}(q, r, W \cup Z) \wedge (p, D \cup E) \in \mathcal{F}[\![P]\!]e \\
& \qquad \wedge (q, U \cup V) \in \mathcal{F}[\![Q]\!]e \wedge (r, U \cup M) \in \mathcal{F}[\![R]\!]e \,\} \\
& \cup \{\, (u, D \cup E \cup F) \mid \exists p, x : D \subseteq \overline{W} \cap \overline{X} \cap \overline{Y} \wedge E, F \subseteq W \cup Z \\
& \qquad \wedge u \in \mathsf{merge}(p, x, W \cup X \cup Y) \wedge (p, D \cup E) \in \mathcal{F}[\![P]\!]e \\
& \qquad \wedge x \in \mathcal{D}[\![Q \|_{W \cup Z} R]\!]e \wedge D \cup F \in \mathbb{P}(\Sigma) \,\}
\end{aligned}$$

Again the second set can be ignored because it is included in the previously discarded failure set. The failure set for the *rhs* can be similarly expanded.

$$\begin{aligned}
& \mathcal{F}[\![R \|_{W \cup Y \cup Z} (P \|_{W \cup X} Q)]\!]e \\
= \; & \{\, (u, D' \cup E' \cup F') \mid \exists p, q, r, z : D' \subseteq \overline{W} \cap \overline{Y} \cap \overline{Z} \wedge E', F' \subseteq W \cup Y \cup Z \\
& \qquad \wedge D' \cup F' = U' \cup V' \cup M' \wedge U' \subseteq \overline{W} \cap \overline{X} \\
& \qquad \wedge V', M' \subseteq W \cup X \wedge u \in \mathsf{merge}(p, z, W \cup Y \cup Z) \\
& \qquad \wedge z \in \mathsf{merge}(q, r, W \cup X) \wedge (p, D' \cup E') \in \mathcal{F}[\![P]\!]e \\
& \qquad \wedge (q, U' \cup V') \in \mathcal{F}[\![Q]\!]e \wedge (r, U' \cup M') \in \mathcal{F}[\![R]\!]e \,\}
\end{aligned}$$

Now both sides have been completely expanded we can proceed with the proof.

(1) $(s, G) \in \mathcal{F}[\![P \|_{W \cup X \cup Y} (Q \|_{W \cup Z} R)]\!]e$ Assumption

(2) $\exists p, q, r, x, D, E, F, U, V, M : D \subseteq \overline{W} \cap \overline{X} \cap \overline{Y}$
$\wedge E, F \subseteq W \cup X \cup Y \wedge U \subseteq \overline{W} \cap \overline{Z}$
$\wedge V, M \subseteq W \cup Z \wedge D \cup F = U \cup V \cup M$
$\wedge G = D \cup E \cup F \wedge s \in \mathsf{merge}(p, x, W \cup X \cup Y)$
$\wedge x \in \mathsf{merge}(q, r, W \cup Z) \wedge (p, D \cup E) \in \mathcal{F}[\![P]\!]e$
$\wedge (q, U \cup V) \in \mathcal{F}[\![Q]\!]e \wedge (r, U \cup M) \in \mathcal{F}[\![R]\!]e$

(3) $\exists p, q, r, z, D, E, F, U, V, M : D \subseteq \overline{W} \cap \overline{X} \cap \overline{Y}$ (merge-Ass)
$\wedge E, F \subseteq W \cup X \cup Y \wedge U \subseteq \overline{W} \cap \overline{Z}$
$\wedge V, M \subseteq W \cup Z \wedge D \cup F = U \cup V \cup M$
$\wedge G = D \cup E \cup F \wedge s \in \mathsf{merge}(r, z, W \cup Y \cup Z)$
$\wedge z \in \mathsf{merge}(p, q, W \cup X) \wedge (p, D \cup E) \in \mathcal{F}[\![P]\!]e$
$\wedge (q, U \cup V) \in \mathcal{F}[\![Q]\!]e \wedge (r, U \cup M) \in \mathcal{F}[\![R]\!]e$

Since we have shown the existence of the traces required to satisfy the *rhs*, traces can be ignored from now on. Therefore, all that remains to be shown is the existence of the failure sets needed to satisfy the *rhs*, i.e. D', E', F', U', V' and M'. These can be defined in terms of the failure sets of the *lhs* D, E, F, U, V and M since the traces are the same for both the *lhs* and the *rhs*. So we define the required failure sets as follows:

$$\begin{aligned}
U' &= ((U \cup V) \cap \overline{W} \cap \overline{X} \cap \overline{Y}) \cup (V \cap \overline{W} \cap \overline{X} \cap Y \cap Z) \cup (E \cap \overline{W} \cap \overline{X} \cap Y) \\
V' &= E \cap (W \cup X) \\
M' &= (U \cup V) \cap (W \cup X) \\
D' &= (U \cap \overline{W} \cap \overline{Y} \cap \overline{Z}) \cup (E \cap \overline{W} \cap \overline{Y} \cap \overline{Z} \cap X) \\
E' &= (U \cap \overline{W} \cap \overline{Z} \cap Y) \cup M \\
F' &= \quad (U \cap X \cap Y) \cup (V \cap (W \cup (X \cap Y) \cup (X \cap Z))) \cup (V \cap \overline{W} \cap \overline{X} \cap Z) \\
&\quad \cup (E \cap (W \cup (X \cap Y) \cup (X \cap Z))) \cup (E \cap \overline{W} \cap \overline{X} \cap Y)
\end{aligned}$$

It is clear from the definitions of these sets that they satisfy the *rhs* requirements.

$$G = D' \cup E' \cup F' \ \wedge \ D' \subseteq \overline{W} \cap \overline{Y} \cap \overline{Z} \ \wedge \ E', F' \subseteq W \cup Y \cup Z$$
$$\wedge \ U' \subseteq \overline{W} \cap \overline{X} \ \wedge \ V', M' \subseteq W \cup X \ \wedge \ D' \cup F' = U' \cup V' \cup M'$$

Proof is trivial and is omitted. Now all that remains to be shown is that these sets are in fact refused by the relevant processes, that is prove each of the following:

$$(p, U' \cup V') \in \mathcal{F}[\![P]\!]e \qquad (q, U' \cup M') \in \mathcal{F}[\![P]\!]e \qquad (r, D' \cup E') \in \mathcal{F}[\![R]\!]e$$

The proof of $(p, U' \cup V') \in \mathcal{F}[\![P]\!]e$ is as follows:

(1)	$(p, D \cup E) \in \mathcal{F}[\![P]\!]e$	Assumption
(2)	$(U \cup V) \cap \overline{W} \cap \overline{X} \cap \overline{Y} \subseteq D$	def. D
(3)	$(E \cap \overline{W} \cap \overline{X} \cap Y) \cup (E \cap (W \cup X)) \subseteq E$	def. E
(4)	$(p, ((U \cup V) \cap \overline{W} \cap \overline{X} \cap \overline{Y}) \cup (E \cap \overline{W} \cap \overline{X} \cap Y)$ $\cup (E \cap (W \cup X))) \in \mathcal{F}[\![P]\!]e$	N3(P), (1), (2), (3)
(5)	$(V \cap \overline{W} \cap \overline{X} \cap Y \cap Z) \subseteq Z \subseteq \overline{A}$	def. Z
(6)	$\forall a \in (V \cap \overline{W} \cap \overline{X} \cap Y \cap Z) : (p^\frown\langle a \rangle, \varnothing) \notin \mathcal{F}[\![P]\!]e$	def. Z
(7)	$(p, U' \cup V') \in \mathcal{F}[\![P]\!]e$	N4(P), (4), (6)

Notes: (5) means that the events in $(V \cap \overline{W} \cap \overline{X} \cap Y \cap Z)$ are outside the alphabet A of P, i.e. are impossible for it. (6) states that in particular they are impossible after P has performed the trace p.

The proofs of $(q, U' \cup M') \in \mathcal{F}[\![Q]\!]e$ and $(r, D' \cup E') \in \mathcal{F}[\![R]\!]e$ are similar, the sets of impossible events for these are $(E \cap \overline{W} \cap \overline{X} \cap Y)$ and $(E \cap \overline{W} \cap \overline{Y} \cap \overline{Z} \cap X)$ respectively.

This concludes the proof of the failure sets equality. Therefore, we have proved that the failure sets and divergence sets of both sides are equal, and hence, the two processes are equivalent. This concludes the proof of the associativity law. □