Concurrency and conflict in CSP

Marta Kwiatkowska
mzk@mcs.le.ac.uk
Department of Mathematics and Computer Science
University of Leicester, Leicester LE1 7RH, United Kingdom
also visiting
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom

Iain Phillips
iccp@doc.ic.ac.uk
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom

Abstract

As part of an effort to give a "truly concurrent" semantics to process algebra, we refine the failures with divergence model for CSP by adding conflict information. We show that most of the CSP laws are preserved, the exception being the expansion law and the idempotency of choice.

1 Introduction

This paper is part of an effort by many authors to give a "truly concurrent" semantics to process algebra. Many authors have previously contributed in this area, see *e.g.* [16, 15, 9, 12, 4, 2, 14], and the methods have included translation into Petri nets, deriving an event structure, and enhancing the labels in labelled transition systems, often leading to descriptions of low-level character. We propose starting at the high level of abstraction usually associated with process algebras, and define a refinement of the existing failures with divergence model directly from the syntax of CSP terms.

As the touchstone for whether a semantics is truly concurrent, as usual we take the example processes $a \,||\, b$ and $a{\cdot}b + b{\cdot}a$, where $||$ denotes parallel composition, $\cdot$ denotes sequential composition and $+$ denotes choice. These two are of course equated in the standard interleaving semantics of process algebras such as CCS, CSP, ACP. However, we would expect them to be distinguished by a truly concurrent semantics, since $a||b$ exhibits concurrency (in that actions a and b are causally independent), whereas $a{\cdot}b + b{\cdot}a$ does not.

We take the aim of this exercise to be not so much to argue that the two processes are not really behaviourally equivalent, but to express in a precise way our understanding that one is "more concurrent" than the other. The ordering that is thus obtained can be used to inform the process of implementation, since implementation has a direction (from specification to implementation, possibly via various refinements) which frequently involves adding extra concurrency by implementing a "black box" as a collection of connected components. Abramsky has made a similar point [1].

Trace theory [13] provides one solution to the example. The sequences of possible actions are just ε, a, b, ab, ba in each case. In the case of $a \parallel b$, since a, b are independent, denoted $a \; \iota \; b$, we say that ab and ba are equivalent, since they are the same except that independent actions are permuted. However, in the case of $a{\cdot}b + b{\cdot}a$ the sequences ab, ba are not equivalent, and so we have distinguished the processes.

The disadvantage of trace theory is that it does not handle nondeterminism and deadlock. For instance the processes $a{\cdot}b$ and $a + a{\cdot}b$ have the same sequences, namely ε, a, ab, and there is no independence between actions in either case. However, the second process may deadlock after performing a, so that a distinction should be made. It is for this reason that Milner was led to consider a semantics for processes more refined than that used in formal language theory. He proposed observation equivalence (bisimulation), which takes us a long way from traces, and is generally considered the most refined behavioural equivalence. Brookes, Hoare and Roscoe [6] took a different approach, in which they refined the traces[1] model with deadlock information telling us which actions may be refused after some sequence of actions, giving the so-called *failures* model.

The aim of this paper is to make concurrency distinctions within CSP by associating to each process P a *conflict* relation $cf(P)$ on the traces of P. Roughly speaking, the more traces of P are in the relation $cf(P)$, the less concurrent P will be. We define $cf(P)$ in Section 4 by induction on the syntax of CSP and then define the "failures with divergence and conflict" *semantics*. We defer discussion of the *model* to Section 6 as this is not yet fully resolved. We define a partial ordering incorporating conflict, which is a natural refinement of the failures–divergence ordering. We also show that the model forms a complete partial order, and that the operators of CSP are monotonic and continuous with respect to the new ordering. The old model can be embedded into the new one by mapping processes onto those with maximal conflict. In Section 5 we find that many of the usual laws hold, but that equivalences involving choice, such as idempotency and the expansion theorem for unravelling the behaviour of a parallel composition, are now refined into inequalities. Finally, in Section 6, the new model and its properties are discussed.

2 An example

Suppose we wish to build a scheduler to control the activation of n processes. The activation of process i is represented by the action a_i. The processes are to start up essentially in cyclic order:

$$a_1, a_2, \ldots, a_n, a_1, \ldots$$

except that we allow the activations to be at most one ahead of their turn. Thus, if it is i's turn then $i+1$ is also allowed to go ahead, but, if it does, then

[1] Note that in the CSP world "trace" means a sequence of actions performed by a process, whereas for trace theory, it means an equivalence class of such sequences under the equivalence generated by the independence relation.

i must go next before it is $i+2$'s turn. The following CSP program achieves this and corresponds fairly directly to the informal specification:

$$P \quad = \quad P_1 \quad \text{where} \quad P_i \quad = \quad a_i P_{i+1} \,\square\, a_{i+1} a_i P_{i+2}.$$

Now suppose we wish to introduce some concurrency into the scheduler. Perhaps the processes are distributed in a ring, and we wish the scheduling to be achieved by passing signals round the ring rather than by the use of a single centralised process. For pure cyclic scheduling it is enough for a_{i+1} to be made dependent on a_i, and the following will achieve the desired result:

$$\begin{array}{rclcrcl} Q & = & Q_1 \,\|\, \cdots \,\|\, Q'_n & \text{where} & Q_i & = & a_i Q'_i \\ & & & & Q'_i & = & a_{i+1} Q_i. \end{array}$$

We adapt this by requiring a_i to be dependent on both a_{i-2} and a_{i-3} (but not a_{i-1}, to allow actions to be one ahead of their turn). We get

$$T \quad = \quad R_1 \,\|\, \cdots \|R_{n-2}\|R'_{n-1}\|R'_n\|S_1\| \cdots \|S'_{n-2}\|S'_{n-1}\|S'_n$$

$$\begin{array}{lrclrcl} \text{where} & R_i & = & a_i R'_i & R'_i & = & a_{i+2} R_i \\ & S_i & = & a_i S'_i & S'_i & = & a_{i+3} S_i. \end{array}$$

We have not indicated the alphabets of the processes, which are the obvious ones. Each event a_i requires the participation of four sub-processes.

One can show that T has the same behaviour as P, and indeed they are equated in the standard semantics of CSP (failures semantics - see Section 3). However, there are differences. T has more concurrency, which is reflected in the fact that T has less conflict (it has the same options, of course). In fact, our example illustrates an extreme opposition, in that P has no concurrency, and T has no choice. In general, we can imagine various programs being equivalent in their observed behaviour, but differing in the amount of concurrency they exhibit. And, as this example shows, more concurrency will tend to be reflected in reduced conflict.

If we wish to reflect these differences there are many possibilities. In this paper we investigate an approach which seems to us to be close to the spirit of the standard observational semantics of CSP, while enabling us to make true concurrency distinctions. We refine failures semantics by adding information about which traces (sequences of actions) are in conflict, that is, cannot occur in the same run. In our semantics T will have less conflict than P (in fact T has none), reflecting the greater concurrency in T.

3 Preliminaries

3.1 Notation

We shall use the following notation conventions throughout the paper. *Act* is a set of action symbols[2]. The set of finite strings (also called *traces*) over the set *Act* is denoted by $Act^\star$; ε is the empty trace. We use a, b, c to range over *Act*,

[2] We prefer to use the term "action" instead of "event" so as to distinguish between actions and their occurrences; the latter are usually referred to as "events".

and s, t, u to range over $Act^\star$. Given $a \in Act$ and traces $s, t \in Act^\star$, we write as, sa and st for the respective concatenations. String prefix order is denoted by $\leq$. We say s *perm* t iff the trace s is a permutation of t. $s \subseteq t$ denotes the multiset order; more formally, $s \subseteq t$ iff $\exists s'.s \; perm \; s' \leq t$. Given a set L, the set of all finite subsets of L will be denoted by $\mathcal{P}_f(L)$, and the set of all its subsets by $\mathcal{P}(L)$. We use A, B, X, Y to range over $\mathcal{P}_f(Act)$. $s \backslash d$, where $d \in Act$, denotes *hiding*, *i.e.* the string s with all occurrences of d deleted. Given $A \in \mathcal{P}_f(Act)$, $s \lceil A$ denotes *restriction*, *i.e.* the string s with all occurrences of actions outside the set A deleted.

We shall also require the following operation of *leftmost difference* on $Act^\star$. Let $a \in Act, s, t \in Act^\star$. Define

$$\begin{array}{lcl} \mathrm{ld}(\varepsilon, s) & = & \text{undefined} \\ \mathrm{ld}(as, t) & = & \begin{cases} \mathrm{ld}(s, t-a) & \text{if } a \in t \\ a & \text{otherwise} \end{cases} \end{array}$$

where $t - a$ denotes the string t with the leftmost occurrence of a deleted.

3.2 The language of CSP

We use a subset of Communicating Sequential Processes (CSP) as described in [7]. This is essentially an alphabet-free version of CSP as introduced in [10]. In this paper only a summary of the language and the model is presented. We refer the reader to [10, 6, 7] for more details.

The syntax of CSP process expressions is as follows:

$$P ::= 0 \mid a{\cdot}P \mid P \sqcap Q \mid P \,\Box\, Q \mid P_A \parallel Q_B \mid P \backslash a \mid f(P) \mid x \mid \mu x.P$$

where P, Q ranges over (syntactic) process expressions, a is an action ranging over the set Act, A, B are (finite) sets of actions, and x is a process variable. 0 represents a deadlocked process (usually denoted STOP). $a{\cdot}P$ denotes prefixing the process P with the action a (usually $a{\rightarrow}P$). We shall often elide $\cdot$ and 0; *e.g.* the process $a{\cdot}b{\cdot}0$ will simply be denoted by ab. $P \sqcap Q$ is the nondeterministic choice (internal), while $P \,\Box\, Q$ the deterministic choice (external). $P_A \parallel P_B$ is the mixed parallel operator; here A is the named alphabet for the process P, B the named alphabet for the process Q, and $P_A \parallel Q_B$ requires simultaneous participation of P and Q on any actions belonging to $A \cap B$, but the process P may progress independently on actions belonging to $A \setminus B$ (and symmetrically for Q). If it is clear from the context, the alphabets will be omitted. $P \backslash a$ denotes the hiding operator, while $f(P)$ renaming. We assume that $f : Act \longrightarrow Act$ is a 1-1 function. Finally, recursion is handled by terms $\mu x.P$, where μ is a variable binding operator.

We shall denote the set of terms defined by the above syntax by CSP. A term is said to be *closed* iff it has no free process variables.

4 Semantics

4.1 "Failures with divergence" semantics

The failures semantic model [6] uses a representation of a process as a set of failures, that is pairs (s, X), where $s \in Act^\star$ is a finite sequence of actions and

X is a finite set of actions. The meaning of a failure (s, X) is as follows: the process *may* perform the sequence of actions s, and then *refuse* all the actions in the set X. Formally, a failure set is a subset:

$$F \subseteq Act^\star \times \mathcal{P}_f(Act)$$

satisfying certain axioms which will be discussed in Section 6. For any set F of failures we define

$$tr(F) \quad = \quad \{s \mid \exists X.(s, X) \in F\}.$$

There exists a natural ordering on the failure sets. We say $F_1 \sqsubseteq F_2$ (F_1 is more nondeterministic than F_2) iff $F_1 \supseteq F_2$.

The original failures semantics was not completely satisfactory in that its least element CHAOS was not absorptive. In particular, the law $P \,||\, \mathrm{CHAOS} = \mathrm{CHAOS}$ was not satisfied. The solution introduced in [7] was to consider a more refined semantics which takes account of divergence. A process P is said to *diverge* if it may engage in an unbounded sequence of internal actions, invisible to its environment. An example of such process is $(\mu x.ax)\backslash a$. In order to account for divergence processes are modelled as pairs $\langle F, D\rangle$, where F is a failure set and $D \subseteq Act^\star$ is a divergence set, subject to certain axioms (see Section 6). The set of all such pairs is denoted by $\mathbb{FD}$. Failure–divergence pairs are ordered by:

$$\langle F_1, D_1\rangle \sqsubseteq \langle F_2, D_2\rangle \quad \Longleftrightarrow \quad F_1 \supseteq F_2 \wedge D_1 \supseteq D_2.$$

This means that the process $\langle F_1, D_1\rangle$ is more nondeterministic than $\langle F_2, D_2\rangle$ if it can diverge whenever $\langle F_2, D_2\rangle$ can diverge and fail whenever $\langle F_2, D_2\rangle$ can fail. The order gives rise to a complete partial lattice; the least element, denoted $\perp$, has the failure set $Act^\star \times \mathcal{P}_f(Act)$ and the divergence set $Act^\star$. Clearly, the divergence–free processes are isomorphic to the original failures model. We use $div(\cdot)$ and $fail(\cdot)$ to extract the components of the pair $\langle F, D\rangle$.

CSP can be given denotational semantics by means of a compositional mapping from process expressions to the failure–divergence pairs [7]. The closed process expressions can be identified with elements of the domain $\mathbb{FD}$, while process expressions containing free variables create the need for environments. An *environment* ρ ranging over the set Env is a function from process variables to failure–divergence pairs $\mathbb{FD}$. Formally, we have a map

$$\mathcal{FD}[\![\cdot]\!] : \mathrm{CSP} \longrightarrow [Env \longrightarrow \mathbb{FD}].$$

It is often convenient to view $\mathcal{FD}[\![\cdot]\!]$ as a pair of functions such that:

$$\mathcal{FD}[\![P]\!]\rho \quad = \quad \langle fail[\![P]\!]\rho,\, div[\![P]\!]\rho\rangle$$

for any process expression P and environment ρ.

The maps $fail[\![\cdot]\!]$ and $div[\![\cdot]\!]$ are defined by induction on the syntax of CSP process expressions. For expressions not involving recursion this is summarised in Tables 1 and 2. As the operations thus defined are continuous, the meaning of recursive process terms can be given by:

$$\mathcal{FD}[\![\mu x.P]\!]\rho \quad = \quad \mu(\lambda\langle F, D\rangle.(\mathcal{FD}[\![P]\!])(\rho + [x \mapsto \langle F, D\rangle])).$$

where μ denotes the least fixpoint of the operator on $\mathbb{FD}$ (shown in brackets) which maps the pair $\langle F, D\rangle$ to the meaning of P in the environment which agrees with ρ, except that it maps the variable x to $\langle F, D\rangle$.

Table 1: The failures table. $\mathrm{C}(X,Y,Z)$ denotes $X \subseteq A, Y \subseteq B, Z \cap (A \cup B) = \emptyset$.

$$\frac{s \in div(P)}{(st, X) \in fail(P)} \qquad \frac{}{(\varepsilon, \emptyset) \in fail(0)}$$

$$\frac{a \notin X}{(\varepsilon, X) \in fail(aP)} \qquad \frac{(s, X) \in fail(P)}{(as, X) \in fail(aP)}$$

$$\frac{(s, X) \in fail(P)}{(s, X) \in fail(P \sqcap Q)} \qquad \frac{(s, X) \in fail(P)}{(s, X) \in fail(Q \sqcap P)}$$

$$\frac{(s, X) \in fail(P)}{(s, X) \in fail(P \,\square\, Q)}\ s \neq \varepsilon \qquad \frac{(s, X) \in fail(P)}{(s, X) \in fail(Q \,\square\, P)}\ s \neq \varepsilon$$

$$\frac{(\varepsilon, X) \in fail(P) \quad (\varepsilon, X) \in fail(Q)}{(\varepsilon, X) \in fail(P \,\square\, Q)}$$

$$\frac{(s, X \cup \{a\}) \in fail(P)}{(s \backslash a, X) \in fail(P \backslash a)} \qquad \frac{(s, X) \in fail(P)}{(f(s), f(X)) \in fail(f(P))}$$

$$\frac{(s \restriction A, X) \in fail(P) \quad (s \restriction B, Y) \in fail(Q) \quad \mathrm{C}(X,Y,Z)}{(s, X \cup Y \cup Z) \in fail(P_A \parallel Q_B)}\ s \in (A \cup B)^\star$$

Table 2: The divergence table

$$\frac{s \in div(P)}{st \in div(P)} \qquad \frac{s \in div(P)}{as \in div(aP)}$$

$$\frac{s \in div(P)}{s \backslash a \in div(P \backslash a)} \qquad \frac{sa^n \in div(P) \ \ \forall n}{s \backslash a \in div(P \backslash a)}$$

$$\frac{s \in div(P)}{f(s) \in div(f(P))}$$

$$\frac{s \restriction A \in div(P) \quad s \restriction B \in tr(Q)}{s \in div(P_A \parallel Q_B)}\ s \in (A \cup B)^\star$$

$$\frac{s \restriction A \in tr(P) \quad s \restriction B \in div(Q)}{s \in div(P_A \parallel Q_B)}\ s \in (A \cup B)^\star$$

$$\frac{s \in div(P)}{s \in div(P \sqcap Q)} \qquad \frac{s \in div(Q)}{s \in div(P \sqcap Q)}$$

$$\frac{s \in div(P)}{s \in div(P \,\square\, Q)} \qquad \frac{s \in div(Q)}{s \in div(P \,\square\, Q)}$$

4.2 The conflict relation

CSP is a specification–oriented language. The point of interest is what can be *guaranteed* of processes in terms of liveness and convergence. As a matter of mathematical convenience, the failures with divergence model records *possible* negative behaviour, *i.e.* failures (deadlocks) and divergence. $P \sqsubseteq Q$ should mean that Q is "better" than P, in the sense that Q makes more guarantees than P. Within the model this becomes $Q \subseteq P$, *i.e.* Q has less "bad" behaviour. As we are interested in refining this model, it is reasonable to assume that the new ordering $P \preceq Q$ should mean that $P \sqsubseteq Q$ and Q guarantees more concurrency. In keeping with the CSP treatment of refusals and divergence, we record *possible* conflicts. The basic idea is that two traces of a process P are in conflict if they *may* belong to different "runs" of P. Clearly, if two traces are not in conflict then they are *guaranteed* to belong to the same run.

To make our notion of possible conflict clearer, consider the following explanation in terms of event structures [17]. Process algebra has been modelled in event structures by various authors. CCS has been treated in [16, 4]. A prime event structure for CSP has been defined in [12].

To each process P we assign a set $E(P)$ of events. Notice that some of these will be labelled with the same action, so that for instance $E(a \,\Box\, a)$ will have two events, both labelled with a. We also record whether these events are hidden, so that for instance $E((a \,\Box\, b)\backslash b)$ will have two events, one of which is hidden. We have a partial ordering $\leq$ on $E(P)$, and also a conflict relation $\#$ which is symmetric. For example, in $E(ab \,\Box\, c)$, $e_1 \leq e_2$ and $e_1 \# e_3$ where e_1, e_2, e_3 are labelled with a, b, c respectively.

A run r of P is defined to be a lower-closed, conflict-free subset of $E(P)$. In other words, if $e \in r$ and $e' \leq e$ then $e' \in r$, and if $e, e' \in r$ then it is not the case that $e \# e'$. Let $run(P)$ denote the set of all runs of P. We then define two runs to be compatible if they have a common extension: r_1 is compatible with r_2 iff there exists r such that $r_1, r_2 \subseteq r$. If they are not compatible, we say they conflict: $r_1 \; cf \; r_2$.

Next we suppose an *observation relation* between traces and runs: $obs(t, r)$ means that t is a possible observation of r. In general, there will be several runs corresponding to a given trace because of different events having the same labels, and of hiding. Also, there will be several traces corresponding to a given run (corresponding to different linearizations of the partial order). Finally, we can obtain the conflict relation on traces:

$$s_1 \; cf(P) \; s_2 \quad \Longleftrightarrow \quad \exists r_1, r_2.\; obs(s_1, r_1) \wedge obs(s_2, r_2) \wedge (r_1 \; cf \; r_2).$$

Instead of actually carrying this out, we prefer to define the conflict relation $cf(P) \subseteq (tr(P))^2$ by induction over syntax (see Table 3). We include the rules (div) and (sym) for reasons of convenience, even though they are not strictly over syntax. Both (div) and (sym) could be replaced by a more elaborate version of the other rules. There are two essential rules, (L-R-$\sqcap$) and (L-R-$\Box$), shown below:

$$\frac{s \in tr(P) \quad s' \in tr(Q)}{s \;\; cf(P \sqcap Q) \;\; s'} \qquad \frac{s \in tr(P) \quad s' \in tr(Q)}{s \;\; cf(P \,\Box\, Q) \;\; s'} \; s, s' \neq \varepsilon$$

The rule (L-R-$\sqcap$) ensures that all traces of P, including the empty trace ε, are in conflict with each trace of Q. On the other hand, according to (L-R-$\Box$) only

Table 3: The conflict relation

(div)	$\dfrac{s \in div(P)}{s\ \ cf(P)\ \ s'}$	(sym)	$\dfrac{s\ \ cf(P)\ \ s'}{s'\ \ cf(P)\ \ s}$
(pref)	$\dfrac{s\ \ cf(P)\ \ s'}{as\ \ cf(a \cdot P)\ \ as'}$		
(hide)	$\dfrac{s\ \ cf(P)\ \ s'}{s \backslash a\ \ cf(P)\ \ s' \backslash a}$	(ren)	$\dfrac{s\ \ cf(P)\ \ s'}{f(s)\ \ cf(f(P))\ \ f(s')}$
(L-$\Vert$)	$\dfrac{s \restriction A\ \ cf(P)\ \ s' \restriction A}{s\ \ cf(P_A \Vert Q_B)\ \ s'}$	(R-$\Vert$)	$\dfrac{s \restriction B\ \ cf(Q)\ \ s' \restriction B}{s\ \ cf(P_A \Vert Q_B)\ \ s'}$
(L-$\sqcap$)	$\dfrac{s\ \ cf(P)\ \ s'}{s\ \ cf(P \sqcap Q)\ \ s'}$	(R-$\sqcap$)	$\dfrac{s\ \ cf(Q)\ \ s'}{s\ \ cf(P \sqcap Q)\ \ s'}$
(L-R-$\sqcap$)	$\dfrac{s \in tr(P)\ \ s' \in tr(Q)}{s\ \ cf(P \sqcap Q)\ \ s'}$		
(L-$\Box$)	$\dfrac{s\ \ cf(P)\ \ s'}{s\ \ cf(P \Box Q)\ \ s'}$	(R-$\Box$)	$\dfrac{s\ \ cf(Q)\ \ s'}{s\ \ cf(P \Box Q)\ \ s'}$
(L-R-$\Box$)	$\dfrac{s \in tr(P)\ \ s' \in tr(Q)}{s\ \ cf(P \Box Q)\ \ s'}\ s, s' \neq \varepsilon$		

those pairs of traces from P and Q are in conflict which are not simultaneously empty. Of the remaining rules, (sym) imposes symmetry, and the rest preserve conflict through the syntactic structure.

Example 4.1 *(Possible conflict)* Note that $a\ \ cf(a \Box b)\ \ b$, $\varepsilon\ \ cf(a \sqcap b)\ \ \varepsilon$, $\varepsilon\ cf(a \sqcap b)\ b$, $a\ cf((a \Vert b) \sqcap a)\ b$.

The complement $(tr(P))^2 \backslash\ cf(P)$ of the conflict relation will often be referred to as the *compatibility* relation and will be denoted by $\overline{cf}(P)$.

Example 4.2 *(Illustration of compatibility)* Note that $a\ \overline{cf}(ab)\ ab$, $a\ \overline{cf}(a \Vert b)$ ba.

The next examples illustrate the difficulty caused by the presence of hiding.

Example 4.3 Let $Q = (a \Box b) \backslash b$. Then $a \in tr(Q)$, $(\varepsilon, \{a\}) \in fail(Q)$, but $\varepsilon\ \overline{cf}(Q)\ \varepsilon$. Also, $\varepsilon\ cf(Q)\ a$.

Example 4.4 Let Q be the same as in Example 4.3, $R = Q \Box a$. Then $\varepsilon\ \overline{cf}(R)\ \varepsilon$, $(\varepsilon, \{a\}) \notin fail(R)$, but $\varepsilon\ cf(R)\ a$. Also, $a\ cf(R)\ a$.

Example 4.5 Let $P = ab \Box a$, $s = ab$, $t = \varepsilon$. Note that $s\ \overline{cf}(P)\ t$, $s\ \overline{cf}(P)$ $s(t - s)$ but $s\ cf(P)\ a$. A similar case is shown in Example 6.4.

The final two examples exhibit confusion, which is characteristic of an interference between concurrency and choice. These phenomena arise in Petri nets.

Example 4.6 *(Symmetric confusion)* Let $P = (a_{\{a\}} \parallel b_{\{b\}}) \Box c$. Then

$$\begin{aligned} tr(P) &= \{ab, ba, c, \ldots\} \\ cf(P) &= \{(ab, c), (ba, c), \ldots\}. \end{aligned}$$

Observe that either a and b happen, in which case they are independent, or c happens. c is only possible initially, but not after a or b.

Example 4.7 *(Asymmetric confusion)* Let $P = (a \Box c)_{\{a,c\}} \parallel bc_{\{b,c\}}$. Then

$$\begin{aligned} tr(P) &= \{ab, ba, bc, \ldots\} \\ cf(P) &= \{(a, bc), (ab, bc), (ba, bc)\}. \end{aligned}$$

Observe that either a and b happens, in which case they are independent, or b is followed by c, in which case a is prevented from happening.

4.3 "Failures with divergence and conflict" semantics

Formally, the proposed "failures with divergence and conflict" semantics is the set of triples $\langle F, D, C\rangle$ where F is a failure set, D is a divergence set, and $C \subseteq (Act^\star)^2$ is a conflict set, subject to certain axioms to be discussed in Section 6.

The set of all failure–divergence–conflict triples will be denoted $\mathbb{FDC}$. Although the discussion of the precise nature of $\mathbb{FDC}$ is deferred to Section 6, it suffices for our present purposes to postulate that it is a sublattice of the appropriate set ordered by $\supseteq$. It will have directed joins given by set intersection. This will be enough for $\mathbb{FDC}$ to serve as the semantic domain for our new semantics.

We define the order over failure–divergence–conflict triples below.

Definition 4.8 *(Order on failure–divergence–conflict triples)*

$$\langle F_1, D_1, C_1\rangle \preceq \langle F_2, D_2, C_2\rangle \iff F_1 \supseteq F_2 \wedge D_1 \supseteq D_2 \wedge C_1 \supseteq C_2.$$

This means that the higher the process in the ordering, the more concurrency (*i.e.* less conflict) it guarantees. This is in agreement with the failures and divergence components, according to which the higher the process in the ordering, the more determinism and the less divergence it guarantees.

As we said earlier, we postulate $(\mathbb{FDC}, \preceq)$ is a dcpo. It may seem that $\mathbb{FDC}$ has arbitrary meets and directed joins. However, in Section 6.2, when we examine the model in detail, we show that meets need not exist — in particular, the meet of a and b.

We overload the notation by using $cf(\cdot)$ to extract the conflict component of the triple $\langle F, D, C\rangle$.

Example 4.9 *(Concurrency vs choice)* Note that $ab \sqcap ba \preceq ab \Box ba \preceq a \parallel b$, but the converse inequalities do not hold.

We can now introduce the semantic map

$$\mathcal{FDC}[\![\cdot]\!] : \mathrm{CSP} \longrightarrow [Env \longrightarrow \mathbb{FDC}]$$

which is a suitable extension, compositional in style, of the map mentioned in Section 4.1. For convenience, we view $\mathcal{FDC}[\![\cdot]\!]$ as a triple of functions such that:

$$\mathcal{FDC}[\![P]\!]\rho \quad = \quad \langle fail[\![P]\!]\rho, div[\![P]\!]\rho, cf[\![P]\!]\rho \rangle$$

for any process expression P and environment ρ mapping process variables to triples in $\mathbb{FDC}$. The maps $fail[\![\cdot]\!]$ and $div[\![\cdot]\!]$ are as defined in Section 4.1, while the definition of $cf[\![\cdot]\!]$ for finite processes is given inductively in Table 3.

If it is clear from the context, we shall confuse the syntactic and semantic CSP operations.

We can show the following result.

Proposition 4.10 The operations of CSP are monotone and continuous in each argument.

Proof: Monotonicity and continuity of the maps $fail[\![\cdot]\!]$ and $div[\![\cdot]\!]$ is known, see *e.g.* [7]. Continuity of hiding for $cf[\![\cdot]\!]$ follows by an argument similar to that for divergence. The remaining cases are straightforward. □

Proposition 4.10 guarantees the existence of least fixpoint solutions to recursive equations, thus ensuring that process definitions using μ are well defined.

The following relationship between the "old" and the "new" model, reminiscent of the relationship between the failures and failures with divergence model, can be stated. We say that a process P has *maximal* conflict iff $cf(P) = tr(P)^2$. It can be shown that the "old" model is isomorphic to those elements in the "new" model which have maximal conflict. Note that, for any CSP term P, $P \sqcap P$ has maximal conflict. Let $\Psi : \mathbb{FD} \longrightarrow \mathbb{FDC}$ be defined by

$$\Psi(\langle F, D \rangle) \quad = \quad \left\langle F, D, tr(P)^2 \right\rangle.$$

We have the following correspondence.

Lemma 4.11 If $x = \mathcal{FD}[\![P]\!]$ then $\Psi(x) = \mathcal{FDC}[\![P \sqcap P]\!]$.

Proposition 4.12 The elements of $(\mathbb{FDC}, \preceq)$ with maximal conflict are isomorphic to $(\mathbb{FD}, \sqsubseteq)$ via the map Ψ.

Proof:

1. Lift the map Ψ to environments ρ.
2. Show $\mathcal{FDC}[\![P]\!](\Psi\rho) \preceq \Psi(\mathcal{FD}[\![P]\!]\rho)$ for all terms P and environments ρ.
3. Show $\mathcal{FDC}[\![P]\!](\Psi\rho) = \Psi(\mathcal{FD}[\![P]\!]\rho)$ for all terms P and environment ρ such that $cf[\![P]\!](\Psi\rho)$ is maximal.

□

5 The laws

The laws which we have verified as sound (but not complete) are shown in Table 4. As expected, the expansion law is now in the form of an inequality. What was less expected, perhaps, is that the idempotency of choice fails.

Table 4: The laws of CSP with conflict

$$
\begin{array}{rcll}
P \,\Box\, P & \preceq & P & \\
P \,\Box\, Q & = & Q \,\Box\, P & \\
P \,\Box\, (Q \,\Box\, R) & = & (P \,\Box\, Q) \,\Box\, R & \\
P \,\Box\, (Q \sqcap R) & \succeq & (P \,\Box\, Q) \sqcap (P \,\Box\, R) & \\
P \,\Box\, 0 & = & P & \\
a(P \sqcap Q) & = & aP \,\Box\, aQ & \\
P \sqcap Q & \preceq & P & \\
P \sqcap Q & = & Q \sqcap P & \\
P \sqcap (Q \sqcap R) & = & (P \sqcap Q) \sqcap R & \\
P \sqcap P \sqcap P & = & P \sqcap P & \\
P \sqcap Q & \preceq & P \,\Box\, Q & \\
P_A \parallel Q_B & = & Q_B \parallel P_A & \\
P_A \parallel (Q_B \parallel R_C)_{B \cup C} & = & (P_A \parallel Q_B)_{A \cup B} \parallel R_C & \\
P_A \parallel 0_B & = & P_A & \text{if } A \cap B = \emptyset \\
P_A \parallel 0_A & = & 0 & \\
aP_A \parallel bQ_B & \succeq & a(P_A \parallel bQ_B) \,\Box\, b(aP_A \parallel Q_B) & \text{where } A \cap B = \emptyset, \\
 & & & a \in A, b \in B \\
(aP)\backslash a & = & P\backslash a & \\
(aP)\backslash b & = & a(P\backslash b) & \text{if } a \neq b \\
(P \sqcap Q)\backslash a & = & (P\backslash a) \sqcap (Q\backslash a) &
\end{array}
$$

6 Models

The purpose of this section is to discuss the "failures with divergence and conflict model" for CSP. As in the case of the failures–divergence model, which constitutes a sublattice of the set of failure–divergence pairs generated by certain closure conditions (axioms (F1)-(F4) and (D1)-(D2) in Section 6.1), we aim to derive similar axioms for the refined model.

6.1 Failures with divergence model

This section gives an overview of issues pertaining to the failures model, which are syntax independent. Recall that a failure is a pair (s, X) where $s \in Act^\star$ is a finite sequence of actions and X is a finite set of actions.

Let F be a failure set, then F must satisfy the following axioms:

(F1) $(\varepsilon, \emptyset) \in F$;
(F2) $(st, \emptyset) \in F \Rightarrow (s, \emptyset) \in F$;
(F3) $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$;
(F4) $(s, X) \in F \wedge (sc, \emptyset) \notin F \Rightarrow (s, X \cup \{c\}) \in F$.

Axioms (F1), (F2) and (F3) are closure conditions – they ensure that the set of traces is non-empty and prefix closed, and that the refusals at a given trace are closed under inclusion. Axiom (F4) states that an impossible action can be included in the refusal set.

Recall that $\langle F, D\rangle$ is a failure–divergence pair if F is a failure set and $D \subseteq Act^\star$ is a divergence set. The failure–divergence pairs must satisfy:

$$\begin{array}{ll} \text{(D1)} & s \in D \Rightarrow st \in D; \\ \text{(D2)} & s \in D \Rightarrow (st, X) \in F. \end{array}$$

Axiom (D1) ensures that every divergence set is suffix closed, while (D2) states that any divergent trace must be refused.

6.2 The new model

In Section 4.3 we defined a map $\mathcal{FDC}[\![\cdot]\!]$ from the syntax of CSP into the domain $\mathbb{FDC}$. $\mathbb{FDC}$ consists of triples $\langle F, D, C\rangle$, where F is a failure set, D is a divergence set and $C \subseteq (Act^\star)^2$ is a conflict set. However, not every such triple deserves to be regarded as a process. We now discuss what additional axioms they should obey to belong to the model.

In the case of the failures with divergence model we can be confident that the axioms (F), (D) are complete, because (assuming Act is finite) any finite failure–divergence pair is in the range of the semantic map $\mathcal{FD}[\![\cdot]\!]$ [8]. We cannot as yet make an analogous statement. However, we have identified certain properties which seem attractive and natural enough that they should be added to the model.

First, we introduce some basic axioms. Let P be $\langle F, D, C\rangle$; then P must satisfy the following axioms:

$$\begin{array}{ll} \text{(C1)} & cf(P) \subseteq (tr(P))^2; \\ \text{(C2)} & s\ cf(P)\ s' \Rightarrow s'\ cf(P)\ s; \\ \text{(C3)} & s \in div(P) \Rightarrow s\ cf(P)\ t. \end{array}$$

According to the axiom (C1), the conflict relation is defined for valid traces only. Axiom (C2) states that the conflict relation must be symmetric. Axiom (C3) ensures that a divergent trace is in conflict with any trace, thus maintaining the catastrophic character of divergence.

Before we give the remainder of the axioms, we consider two relations which are derived from the conflict relation, or, more precisely, from its complement, the compatibility relation. The relation $co(P)$ is similar to Mazurkiewicz trace equivalence [13], while $pre(P)$ is similar to the trace preorder [11], except that in our model they are not reflexive. Intuitively, Mazurkiewicz trace equivalence equates those traces which are observations of the same run — they may only differ in the order of consecutive independent actions. The trace preorder is a version of string prefix order which ignores the permutations of consecutive independent actions.

Definition 6.1 *(Concurrency)* We say $s\ co(P)\ t$ iff $s\ perm\ t$ and $s\ \overline{cf}(P)\ t$.

Definition 6.2 *(Precedence)* We say $s\ pre(P)\ t$ iff $s \subseteq t$ and $s\ \overline{cf}(P)\ t$.

Note that $s\ co(P)\ t$ iff $s\ pre(P)\ t$ and $t\ pre(P)\ s$.

The following example illustrates $co(P)$.

Example 6.3 Let $P = (a\|b)\Box ab$. Then $ab\ perm\ ba$, $ab\ cf(P)\ ab$, $ab\ cf(P)\ ba$, $ba\ co(P)\ ba$.

The relation $co(P)$ equates those traces which are observations of *the same* run, while the relation $pre(P)$ relates an observation of an initial segment of some run with (an observation of) the run itself. It is then reasonable to expect that if $s\ co(P)\ t$ then the behaviour of the system after s should be identical with that after t. We therefore propose the following further axioms:

(C4) $(su, X) \in fail(P), s\ co(P)\ t \Rightarrow (tu, X) \in fail(P)$;
(C5) $u\ cf(P)\ sv, s\ co(P)\ t \Rightarrow u\ cf(P)\ tv$;
(H1) $s\ pre(P)\ t\ pre(P)\ u \Rightarrow s\ pre(P)\ u$;

We expect to see some connection between refusal sets and conflict. For instance, we cannot have both $(sa, \emptyset)$ and $(s, \{a\})$ as observations of a single run. Similarly, if we observe $(s, \{a\})$ and $(s, \{b\})$, and if these are to belong to the same run, then we must also be able to observe $(s, \{a, b\})$. We therefore propose two further axioms. Recall that $\mathrm{ld}(s, t)$ denotes the leftmost difference of traces s and t.

(C6) $(s, \{a\}) \in fail(P),\ t \in tr(P), a = \mathrm{ld}(s, t) \Rightarrow s\ cf(P)\ t$;
(C7) $(s, X) \in fail(P), (s, Y) \in fail(P), s\ \overline{cf}(P)\ s \Rightarrow (s, X \cup Y) \in fail(P)$;

If we observe sa and sb, and these do not conflict, then we expect also to observe sab and sba. This is the forward diamond property as seen in asynchronous transition systems [15] and elsewhere. This holds for CSP processes in our model. In fact, if P is constructed from 0, prefixing, $||$ and $\backslash$ (so that P is conflict-free), then we can also show $sab\ co(P)\ sba$. However, as shown in Example 6.4, the stronger conclusion does not hold in general.

Example 6.4 *(Weak diamond property)* Let $P = Q \backslash d$ where

$$Q \quad = \quad ad(d \,\square\, d)_{\{a,d\}} \mid\mid bdd_{\{b,d\}}.$$

Then $a\ \overline{cf}(P)\ b, a\ \overline{cf}(P)\ a, b\ \overline{cf}(P)\ b$. However, $ab\ cf(P)\ ba$.

Unfortunately, when we come to formulate an axiom embodying the forward diamond property, we find the position is more complicated. Suppose that we observe ab and cd, and these do not conflict. Then we should also be able to observe $acbd$, $cabd$ and any other combinations so long as a comes before b and c comes before d. What is happening is that we have inferred a partial ordering on the four events a, b, c, d.

$$\begin{tikzcd} a \arrow[d] & c \arrow[d] \\ b & d \end{tikzcd}$$

Any trace which is a linearization of the partial ordering should be observable. Unfortunately, this is still not the full picture. If we are given two traces which conflict, we cannot expect to deduce anything further about the process. However, consider this example: a process has traces abc, ba, bda. Suppose also that $abc\ cf(P)\ bda$, but that the other two pairs do not conflict. Then a and b in abc and bda apparently could belong to different runs. However, because neither conflicts with ba we can infer that they are in fact from the same run. It is still the case however that c and d conflict. We therefore deduce the following event structure:

$$\begin{tikzcd}
a \arrow[dr] & & b \arrow[dl] & \\
& c \arrow[rr, dotted, no head] & & d
\end{tikzcd}$$

where the dotted line denotes conflict. A possible trace based on this would be abd, which we could not deduce from considering ba, bda on their own.

We are led to the following hypothesis. If T is a set of traces of a process P, then there is an event structure E associated with T (and $cf(P)$ on T). Let $con(T, s)$ hold if s is a possible configuration of E. Then our next axiom states

$$\text{(C8)} \quad T \subseteq tr(P), con(T, s) \Rightarrow s \in tr(P).$$

We earlier postulated that $\mathbb{FDC}$ should be a dcpo. The following result shows that $\mathbb{FDC}$ does indeed possess arbitrary directed joins given by set intersection, even when all the axioms we have considered are admitted.

Proposition 6.5 Axioms (C1) – (C8) and (H1) are preserved by directed set intersection.

Proof: Omitted. $\square$

The following is a conjecture, which we have proved for all cases except for hiding and axiom (H1).

Conjecture 6.6 The above axioms are preserved by all the CSP operations.

The following example shows that meets do not exist in $\mathbb{FDC}$, unlike in $\mathbb{FD}$, which is a complete semi-lattice with arbitrary meets corresponding to set union.

Example 6.7 *(Non-existence of meets)* Let $P_1 = a \sqcap b, P_2 = (a \Box b \Box c) \backslash c$. Their conflict sets are as follows:

$$\begin{aligned} cf(P_1) &= \{(\varepsilon, \varepsilon), (\varepsilon, a), (b, \varepsilon), (a, b)\} \\ cf(P_2) &= \{(\varepsilon, a), (b, \varepsilon), (a, b)\} \end{aligned}$$

so $cf(P_1) \supseteq cf(P_2)$. However, $fail(P_1) \subseteq fail(P_2)$ since $fail(P_1) = fail(a) \cup fail(b)$, while $fail(P_2) = fail(P_1) \cup \{(\varepsilon, \{a, b\})\}$. Thus, $P_1 \preceq a, b$ and $P_2 \preceq a, b$, but P_1 and P_2 are incomparable. It is easily seen that $cf(P_1)$ and $cf(P_2)$ are least with respect to their respective failure sets. We can now show that the meet of a and b does not exist. Suppose Q is the meet of a and b, then $fail(Q) \subseteq fail(P_1)$, $fail(a), fail(b) \subseteq Q$. Hence, $fail(Q) = fail(P_1)$. Also, $cf(Q) \subseteq cf(P_1)$. But $cf(P_1)$ is least with respect to $fail(P_1)$. Hence, $Q = P_1$. But then $cf(Q) \not\subseteq cf(P_2)$, and $P_2 \preceq a, b$. Therefore, Q is not above P_2 and Q cannot be the required meet.

7 Further work

Obtaining a complete set of axioms for the failures–divergence–conflict triples has proved more difficult than originally thought. One reason for this state

Table 5: The causality relation

(div)	$\dfrac{s \in div(P)}{s \;\; cs(P) \;\; s'}$		
(init)	$\dfrac{s \in tr(P)}{a \;\; cs(a{\cdot}P) \;\; as} s \neq \varepsilon$	(pref)	$\dfrac{s \;\; cs(P) \;\; s'}{as \;\; cs(a{\cdot}P) \;\; as'}$
(hide)	$\dfrac{s \;\; cs(P) \;\; s'}{s \backslash a \;\; cs(P \backslash a) \;\; s' \backslash a}$	(ren)	$\dfrac{s \;\; cs(P) \;\; s'}{f(s) \;\; cs(f(P)) \;\; f(s')}$
($\parallel$)	$\dfrac{s \restriction A \;\; cs(P) \;\; s' \restriction A \qquad s \restriction B \;\; cs(Q) \;\; s' \restriction B}{s \;\; cs(P_A \parallel Q_B) \;\; s'}$		
(L-$\sqcap$)	$\dfrac{s \;\; cs(P) \;\; s'}{s \;\; cs(P \sqcap Q) \;\; s'}$	(R-$\sqcap$)	$\dfrac{s \;\; cs(Q) \;\; s'}{s \;\; cs(P \sqcap Q) \;\; s'}$
(L-$\Box$)	$\dfrac{s \;\; cs(P) \;\; s'}{s \;\; cs(P \Box Q) \;\; s'}$	(R-$\Box$)	$\dfrac{s \;\; cs(Q) \;\; s'}{s \;\; cs(P \Box Q) \;\; s'}$

of affairs is that, perhaps due to the presence of hiding, the conflict relation on traces, or even failures, is not sufficiently refined to deal with the hidden events. A possible refinement, sufficient to defeat the counter-example to one of the versions of the diamond property (Example 6.4) would be to add causality information. This can be done in terms of syntax-directed rules similar to the conflict relation. A candidate definition has been summarized in Table 5. The essential rules are (init) and (pref), with the remaining rules preserving causality relationships through syntactic structure. Then we could derive a *conflict or causality* relation by putting:

$$cc(P) \quad = \quad cf(P) \cup cs(P) \cup cs(P)^{-1} \; .$$

The relation $cc(P)$, which is symmetric, would replace the relation $cf(P)$ in the formal triple $\langle F, D, C \rangle$.

Another possibility is to adopt a more discriminating notion of an observation, such as failures, instead of traces. Observations as failures are well motivated, since they can be explained in terms of experiments on machines. A generalization of the conflict relation to work on failures instead of traces is straightforward, and it would lead to finer distinctions. Preliminary analysis shows that processes P_1 and P_2 in Example 6.7 become incomparable, but we cannot yet comment on the importance of this fact.

A third possibility is to develop a two-level framework, which would have some behavioural model with runs at the lower level, and a mapping relating runs to (possibly different) notions of observations. The lower-level could, for example, be an event structure derived from syntax by the rules similar to the construction of Boudol and Castellani for CCS [5]. The behavioural phenomena present at the lower level would then induce the corresponding phenomena on observations. It would appear, however, that this approach might take us further away from our intention to work at an abstract level.

Finally, we would like to derive a complete axiomatization of the failure–divergence–conflict triples, analogous to the axioms (F1)-(F4) and (D1)-(D2). It is relatively well known that any finite CSP process is equivalent to one obtained using 0, prefixing, $\sqcap$ and $\Box$ [8]. The interesting point is that the parallel operator $||$ is not used. We would anticipate that in our model, if there is a satisfactory axiomatization, the parallel operator would play an essential role.

8 Conclusion

We have presented a refinement of the standard failures with divergence model for CSP, the purpose of which was to weaken the expansion law to an inequality. We have shown that the additional information about concurrency, conflict and causality can be derived systematically from the syntax of processes, without the need for additional labelling of actions, as long as the relations are interpreted as representing the possibility (or, dually, a guarantee) of the given phenomenon. We have carried out the construction for a large subset of CSP, which includes hiding. We have found that most equalities hold true in the refined model, the exceptions being the expansion law and the idempotency of choice, which have now taken the form of inequalities. The resulting inequational calculus is only slightly more complex than the original failures with divergence calculus.

We believe we have not yet reached a satisfactory characterization of the elements of the model $(\mathbb{FDC}, \preceq)$. This is probably due to a combination of factors, including the difficulty caused by hiding, the need to axiomatize the possibility of conflict, and the fact that working at the level of observations means that we are in effect working with inexact descriptions of the behaviour, since the runs of the system cannot be known. However, a number of pleasing facts are already clear, such as the correspondence of the "old" model to the elements in the "new" model with maximal conflict.

We believe that this framework is, to a large extent, independent of the choice of the language and model. However, we should point out that it is the semantics of the CSP parallel that makes the construction workable.

Acknowledgements

We would like to acknowledge Samson Abramsky and Mike Shields for discussions on the subject. The first author would like to acknowledge the Nuffield Science Foundation grant (SCI/124/528/G) for providing funds to visit the Theory and Formal Methods Section of the Department of Computing, Imperial College. The second author would like to acknowledge the support of the SERC grant GR/F72475.

References

[1] Samson Abramsky. Causal semantics in process algebra. Draft, 1989.

[2] Luca Aceto and Uffe Engberg. Failure semantics for a simple process language with refinement. In *Foundations of Software Technology and Theoretical Computer Science*, pages 89–108. Springer-Verlag, 1991.

[3] S. Blamey. The soundness and completeness of axioms for CSP processes. In G.M. Reed, A.W. Roscoe, and R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 29–56. Oxford University Press, 1991.

[4] Gerard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer-Verlag, 1989.

[5] Gerard Boudol and Ilaria Castellani. Three equivalent semantics for CCS. In *Semantics of Systems for Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[6] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[7] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer-Verlag, 1985.

[8] Stephen Brookes. *A Model for Communicating Sequential Processes*. PhD thesis, 1983.

[9] P. Degano and U. Montanari. Concurrent histories: A basis for observing distributed systems. *Journal of Computer and System Sciences*, 34:422–461, 1987.

[10] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[11] Marta Z. Kwiatkowska. On the domain of traces and sequential composition. In S. Abramsky and T.S.E. Maibaum, editors, *TAPSOFT'91*, volume 493 of *Lecture Notes in Computer Science*, pages 42–56. Springer-Verlag, 1991.

[12] R. Loogen and U. Goltz. A non-interleaving semantics model for nondeterministic concurrent processes. Technical Report 87-15, RWTH Aachen, 1987.

[13] Antoni Mazurkiewicz. Basic notions of trace theory. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1989.

[14] M. Mukund and M. Nielsen. CCS, locations and asynchronous transition systems. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[15] M.W. Shields. Deterministic asynchronous automata. In *Formal Methods in Programming*. North Holland, 1985.

[16] Glynn Winskel. Event structure semantics for CCS and related languages. In *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 561–567. Springer-Verlag, 1982.

[17] Glynn Winskel. Event structures. In *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer-Verlag, 1987.