AN IMPROVED FAILURES MODEL FOR COMMUNICATING PROCESSES

S. D. Brookes
Carnegie-Mellon University
Pittsburgh, Pa.
USA

A. W. Roscoe
Programming Research Group
Oxford University
Oxford
England

0. Abstract.

We extend the failures model of communicating processes to allow a more satisfactory treatment of divergence in addition to deadlock. The relationship between the revised model and the old model is discussed, and we make some connections with various models proposed by other authors.

1. Introduction.

The papers [3,4] introduced the *failure sets* model for communicating sequential processes. This model, an extension of the *traces* model of [13], was able to represent nondeterministic behaviour in a simple but effective way. We showed how to use this model to give a denotational semantics to an abstract version of Hoare's language CSP [14], and used it to prove some theorems about the behaviour of programs. The model enjoyed many elegant mathematical properties, which facilitated formal manipulation and derivation of process properties.

The failures model of processes is able to support a formal treatment of *deadlock* properties. A process is said to *deadlock* if it reaches a stage where it is unable to participate further in events; this property is captured very simply by the failures model, since a potential deadlock corresponds to the ability to refuse all events and this is reflected directly in the structure of the failure set of a process. However, there are problems associated with the treatment in this model of the phenomenon of *divergence*. A process *diverges* when it is engaged in an infinite unbroken sequence of internal actions invisible to its environment, and as a result leaves its environment waiting eternally for a response. It is important to be able to reason about the possibility of divergence, especially when trying to establish a *liveness* property, such as the ability of a process to make some visible response to its environment. The original failures model has certain weaknesses in its treatment of diverging processes, as remarked in [11,12,19,20,2,23], and we will make in this paper suitable alterations to the model which allow a more satisfactory account to be given. The need for these adjustments was originally, independently, suggested by Hennessy and de Nicola in [23] and by Roscoe in [12], and had a direct influence on the development of [2]. The relationship of the new model with the models of Hennessy, de Nicola, and other authors, is discussed in more detail in the final section of this paper. The new model retains the ability to model deadlock, and is thus in a well defined sense an improvement over the original. Again the model possesses an elegant mathematical structure and is well suited to formal manipulation.

A second adjustment, independent of the treatment of divergence, is also suggested in this paper. In the original failures model of processes all refusal sets were finite. This condition is natural when the processes have a finite alphabet of discourse, so that each process is only capable of participation in events drawn from a finite set. However, when processes are allowed to have infinite alphabets it still seems unnatural not to be able to tell explicitly from the semantics of a process whether or not it can refuse an infinite set of events. We will introduce here infinite refusal sets, but with a closure condition making the change from the old model largely cosmetic. In particular, the old model will be seen to "sit inside" the new one in the natural way, so that we are clearly making a reasonable generalization. With this adjustment in place it becomes slightly easier to formulate some of the deadlock properties of processes, since deadlock corresponds explicitly to the refusal of the entire alphabet of events.

We revise the definitions of some of the process compositions from [3,4] to take divergence into account more accurately, and we give examples to show inadequacies in the earlier definitions in their handling of this phenomenon.

Outline.

The first two sections of the paper introduce the old and the new models, and use them to give a semantics to the version of CSP described in [3,4]. Some intuitions are given to justify the changes we have made in building the new model, and the relationship between the old and new models is analysed. The third section gives some examples of applications of the model, defining in the new model the semantics of some interesting forms of process composition which were used in [3,4]. We see that, with care, all of the applications discussed in those papers may be transferred to this more general setting. The same is true of the proof techniques described in the appendix of [4]. The fourth section contains a comparison of the work of this paper with that of other authors, setting our work in a more general context. This section also contains some conclusions and points the way forward to future research.

For obvious reasons the contents of this paper overlaps the material of [3,4] to a substantial degree. In order to avoid too much repetition, and to restrict the length of this paper, much of the material of the earlier papers is assumed.

Notation.

Throughout this paper we will use the following conventions in notation. Given a set Σ of *events*, the set of finite sequences or *traces* over Σ will be denoted Σ^*. We use a, b, c to range over Σ, and s, t, u to range over Σ^*. The empty sequence is $\langle\,\rangle$, and the sequence with elements $a_1, \ldots, a_n$ in that order is written $\langle a_1, \ldots, a_n \rangle$, although sometimes we may omit the braces and write $a_1 \ldots a_n$. Given traces s and t, we write st for their concatenation. We say $s \leq t$ (s is a *prefix* of t) if there is a trace u such that $su = t$; such a trace u is called a *suffix* of t. The powerset of Σ is denoted $\mathcal{P}(\Sigma)$, while we use $p(\Sigma)$ for the finite powerset (the set of all finite subsets); of course, if Σ itself is finite these two powersets coincide. Finally X, Y, Z range over $\mathcal{P}(\Sigma)$ or $p(\Sigma)$, depending on the context.

2. The failures model.

In this section we begin by recalling the earlier *failures model* of [3,4] and its use to give a semantics to a version of CSP.

A process is regarded as an agent which communicates with its environment by performing actions or *events* drawn from an alphabet Σ. Each event can be thought of as an atomic action. Sequential processes are characterised at least partially by the set of possible sequences of events in which they may participate; this constitutes the so-called *trace set* of a process [13]. However, since we are going to be modelling processes with nondeterministic behaviour, traces are not enough. The trace set of a process does not indicate the possibility that deadlock might occur as the result of a nondeterministic decision by a process. The effect of a nondeterministic decision by a process is to restrict its ability to communicate on the next step, by choosing a set of events which will not be possible on that step. Accordingly, the concept of a *failure* suggests itself as a means of modelling the effects of nondeterministic decisions. A failure has the form

$$(s, X),$$

where s is a finite sequence of events and X is a finite set of events. If a particular failure (s, X) is possible for a process, then the process *may* perform the sequence of events s and then be unable to perform any of the events in the set X; we say the process can do s then refuse X.

Bearing in mind the intuition behind this notion of failures, the failure set of a process satisfies four simple conditions (M1)–(M4) below. We therefore define a failure set to be any subset

$$F \subseteq \Sigma^* \times p(\Sigma)$$

satisfying these conditions:

$$(\langle\rangle, \emptyset) \in F \tag{M1}$$
$$(st, \emptyset) \in F \Rightarrow (s, \emptyset) \in F \tag{M2}$$
$$(s, X) \in F \ \&\ Y \subseteq X \ \Rightarrow\ (s, Y) \in F \tag{M3}$$
$$(s, X) \in F \ \&\ (s\langle c\rangle, \emptyset) \notin F \ \Rightarrow\ (s, X \cup \{c\}) \in F. \tag{M4}$$

We will denote the failures model by M.

For any set F of failures we define

$$\begin{aligned} \text{traces}(F) &= \{\, s \mid \exists X.\, (s, X) \in F \,\} \\ \text{initials}(F) &= \{\, c \mid \langle c\rangle \in \text{traces}(F) \,\} \\ \text{refusals}(F) &= \{X \mid (\langle\rangle, X) \in F\} \\ F\, \underline{\text{after}}\, s &= \{\, (t, X) \mid (st, X) \in F \,\}. \end{aligned}$$

By condition (M3), s is a trace of F if and only if $(s, \emptyset)$ is a failure of F. Thus, (M1) and (M2) state that the traces of a process form a non-empty, prefix-closed set. (M2) says that if a process can refuse all events in a set Y then it can also refuse all subsets of Y. Finally, (M4) states that an event which is impossible on the next step can be included in a refusal set. It is easy to see that for any trace s of F, the set $F\, \underline{\text{after}}\, s$ also satisfies the conditions above; this represents the behaviour of the process once it has performed the sequence s.

Since failures represent the results of nondeterministic decisions made by a process, there is a natural partial ordering on failure sets:

$$F_1 \sqsubseteq F_2 \Leftrightarrow F_1 \supseteq F_2.$$

We read $F_1 \sqsubseteq F_2$ as saying that F_1 is more nondeterministic than F_2. The set of all failure sets, ordered by this relation, forms a complete semi-lattice. This means that every non-empty collection of processes has a greatest lower bound (union) and every directed set of processes has a least upper bound (intersection); in particular, the intersection of a chain of processes is again a process. The least element (called CHAOS in [3,4]) is simply $\Sigma^* \times p(\Sigma)$, the set containing all possible failures. The maximal elements, the so-called *deterministic* processes, are characterised by the condition:

$$(s, X) \in F \ \Leftrightarrow\ X \cap \text{initials}(F\, \underline{\text{after}}\, s) = \emptyset,$$

or, equivalently, that their refusal sets contain only impossible events.

CSP operations.

Next we recall the abstract syntax given in [3,4] for a version of CSP. We use P, Q to range over (syntactic) processes. The following BNF-style grammar defines the syntax of our process language:

$$\begin{aligned} P ::= {} & \mathrm{STOP} \mid \mathrm{SKIP} \mid (a \to P) \mid P \sqcap Q \mid P \,\Box\, Q \mid \\ & P \| Q \mid P ||| Q \mid P;Q \mid P \backslash a \mid f^{-1}(P) \mid f(P) \mid p \mid \mu p.P \end{aligned}$$

STOP represents a deadlocked process, and SKIP represents a process which terminates successfully. Two forms of parallel composition are represented by $P \| Q$ and $P|||Q$; the former is known as *synchronous* parallel composition, the latter *asynchronous.* Sequential composition is denoted $P;Q$, and prefixing a single event is denoted $(a \to P)$. Two forms of choice are represented by $P \,\Box\, Q$ and $P \sqcap Q$; the first form is "controllable" and the second "uncontrollable" or purely nondeterministic. The hiding operation $P \backslash a$ conceals all occurrences of the event a. We let f range over a set of *alphabet transformations*; these are renaming functions $f : \Sigma \to \Sigma$ satisfying the *finite pre-image property,* i.e. that for every $a \in \Sigma$ the set $f^{-1}(a) = \{ b \in \Sigma \mid f(b) = a \}$ is finite, so that only finitely many events become identified under a renaming. The process $f^{-1}(P)$ can perform an event b whenever P can perform the image event $f(b)$. Conversely, the process $f(P)$ performs $f(a)$ whenever P can perform a. For further explanation of the nature and significance of these syntactic operations see [4]. In the final two clauses p ranges over a set of process identifiers and $\mu p.P$ is a recursive term, corresponding informally to a recursive process definition of the form $p = P$.

We will denote the set of terms defined by this syntax by TCSP. A term is closed if it has no free process identifiers. In order to interpret terms with free process identifiers we will use an *environment,* which we take to be a function ρ from process identifiers to failure sets. Let MEnv be the set of all environments of this type. We use the notation $\rho + [p \mapsto F]$ for the environment which agrees with ρ except that it maps the process identifier p to the failure set F.

The failures semantics of TCSP is summarized below; these clauses are essentially the same as the definitions given in [3,4], except that we have made explicit use of environments in order to treat the semantics of recursive terms rather more rigorously. In the earlier papers we did not make an explicit distinction between syntax and semantics, preferring rather to blur the distinction and use the same notation for the syntactic as well as semantic operations. Here we are forced to emphasise the separation of syntax and semantics, because we will later have another semantics to discuss.

This semantics is denotational, in that the failure sets of compound processes are definable from the failure sets of their components. We assume familiarity with the basic ideas of denotational semantics, as explained for example in [26]. We define the semantic function

$$\mathcal{M} : \mathrm{TCSP} \to [\mathrm{MEnv} \to M]$$

by the clauses:

$$\begin{aligned}
\mathcal{M}[\![p]\!]\rho &= \rho[\![p]\!] \\
\mathcal{M}[\![\mathrm{STOP}]\!]\rho &= \{ (\langle\rangle, X) \mid X \in p(\Sigma) \} \\
\mathcal{M}[\![\mathrm{SKIP}]\!]\rho &= \{ (\langle\rangle, X) \mid \surd \not\in X \ \&\ X \in p(\Sigma) \} \cup \{ (\surd, X) \mid X \in p(\Sigma) \} \\
\mathcal{M}[\![(a \to P)]\!]\rho &= \{ (\langle\rangle, X) \mid a \not\in X \ \&\ X \in p(\Sigma) \} \cup \{ (\langle a\rangle s, X) \mid (s, X) \in \mathcal{M}[\![P]\!]\rho \} \\
\mathcal{M}[\![P \sqcap Q]\!]\rho &= \mathcal{M}[\![P]\!]\rho \cup \mathcal{M}[\![Q]\!]\rho \\
\mathcal{M}[\![P \Box Q]\!]\rho &= \{ (\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{M}[\![P]\!]\rho \cap \mathcal{M}[\![Q]\!]\rho \} \\
&\quad \cup \{ (s, X) \mid s \neq \langle\rangle \ \&\ (s, X) \in \mathcal{M}[\![P]\!]\rho \cup \mathcal{M}[\![Q]\!]\rho \} \\
\mathcal{M}[\![P \parallel Q]\!]\rho &= \{ (s, X \cup Y) \mid (s, X) \in \mathcal{M}[\![P]\!]\rho \ \&\ (s, Y) \in \mathcal{M}[\![Q]\!]\rho \} \\
\mathcal{M}[\![P|||Q]\!]\rho &= \{ (u, X) \mid \exists s, t.\, u \in \mathrm{merge}(s, t) \ \& \\
&\quad (s, X) \in \mathcal{M}[\![P]\!]\rho \ \&\ (t, X) \in \mathcal{M}[\![Q]\!]\rho \} \\
\mathcal{M}[\![P; Q]\!]\rho &= \{ (s, X) \mid s \text{ tick-free} \ \&\ (s, X \cup \{\surd\}) \in \mathcal{M}[\![P]\!]\rho \} \\
&\quad \cup \{ (st, X) \mid (s\surd, \emptyset) \in \mathcal{M}[\![P]\!]\rho \ \&\ s \text{ tick-free} \ \&\ (t, X) \in \mathcal{M}[\![Q]\!]\rho \} \\
\mathcal{M}[\![P \backslash a]\!]\rho &= \{ (s \backslash a, X) \mid (s, X \cup \{a\}) \in \mathcal{M}[\![P]\!]\rho \} \\
&\quad \cup \{ ((s \backslash a)t, X) \mid \forall n.\, (sa^n, \emptyset) \in \mathcal{M}[\![P]\!]\rho \} \\
\mathcal{M}[\![f^{-1}(P)]\!]\rho &= \{ (s, X) \mid (f(s), f(X)) \in \mathcal{M}[\![P]\!]\rho \} \\
\mathcal{M}[\![f(P)]\!]\rho &= \{ (f(s), X) \mid (s, f^{-1}(X)) \in \mathcal{M}[\![P]\!]\rho \} \\
\mathcal{M}[\![\mu p.P]\!]\rho &= \mathrm{fix}(\lambda F. \mathcal{M}[\![P]\!](\rho + [p \mapsto F])).
\end{aligned}$$

For an open term the environment explicitly supplies a meaning for free identifiers. In the clause for $P|||Q$, we use the notation merge(s, t) for the set of all traces obtained by interleaving the traces s and t. The special event $\surd$ is used to denote successful termination, and is used in a sequential composition $P;\, Q$ to signal the starting point of the second process. A trace is tick-free if it does not contain an occurrence of this event, and a trace ending with $\surd$ represents termination. In the definition of $P\backslash a$, we use the notation $s\backslash a$ for the result of removing all occurrences of a from the trace s. Further explanation and intuitions for these definitions will be found in [4].

As mentioned above, the failures model M forms a complete semi-lattice under the superset ordering. This ordering amounts to a measure of the amount of nondeterminism a process can exhibit. All of the failure set operations used in the semantic clauses above are continuous, so we can appeal to the Knaster-Tarski Fixed Point Theorem to justify the existence of least fixed points [26]. We have used the notation *fix* for the least fixed point operator. Thus, the semantics of a recursively defined process is obtained as the least fixed point of the corresponding function from failures to failures. By continuity, this fixed point can be obtained in the usual way as a limit: the failure set of the process $\mu p.P$ in environment ρ is generated as the intersection of the sequence

$$\begin{aligned} F_0 &= \text{CHAOS}, \\ F_{n+1} &= \mathcal{M}[\![P]\!](\rho + [p \mapsto F_n]) \qquad (n \geq 0). \end{aligned}$$

As an example, the term $\mu p.(a \to p)$ denotes a process which has the ability to perform an unlimited number of a events:

$$\mathcal{M}[\![\mu p.(a \to p)]\!]\rho = \{(a^n, X) \mid n \geq 0 \ \& \ a \notin X \ \& X \in p(\Sigma)\}.$$

This process satisfies the fixed point equation $P = (a \to P)$. The recursion $\mu p.p$ denotes the failure set CHAOS, and is the most nondeterministic of all processes. Mutual recursions can be dealt with in a similar fashion.

We write

$$\begin{aligned} P \sqsubseteq_M Q &\Leftrightarrow \forall \rho.[\mathcal{M}[\![P]\!]\rho \supseteq \mathcal{M}[\![Q]\!]\rho], \\ P \equiv_M Q &\Leftrightarrow \forall \rho.[\mathcal{M}[\![P]\!]\rho = \mathcal{M}[\![Q]\!]\rho]. \end{aligned}$$

In other words, $P \equiv_M Q$ when the two processes have identical failure sets (in this model). When $P \sqsubseteq_M Q$ we say that P is more nondeterministic than Q. This relation induces a pre-ordering on TCSP terms.

In view of the Fixed Point Theorem, a process defined by recursion should satisfy its definition. This is expressed formally in the following way. Let P be a term with free process identifier p. We write $[Q\backslash p]P$ for the term arising by replacing every free occurrence of p in P by Q, with suitable name changes to avoid clashes. Then we have:

$$\mu p.P \equiv_M [(\mu p.P)\backslash p]P.$$

We will often suppress the μ notation and simply define a process by the fixed point equation it is required to satisfy; the implicit understanding when we do this is that we are defining the *least* fixed point.

The following properties of processes will be assumed. Proofs may be found in the earlier papers [3,4] or else in [19,20]. This is not an exhaustive list or a complete set of true equivalences; [19] contains a complete set of axioms for a significant subset of our language (omitting some of the operators).

$$
\begin{aligned}
P \mathbin{\Box} P &\equiv_M P \\
P \mathbin{\Box} Q &\equiv_M Q \mathbin{\Box} P \\
P \mathbin{\Box} (Q \mathbin{\Box} R) &\equiv_M (P \mathbin{\Box} Q) \mathbin{\Box} R \\
P \mathbin{\Box} (Q \sqcap R) &\equiv_M (P \mathbin{\Box} Q) \sqcap (P \mathbin{\Box} R) \\
P \sqcap (Q \mathbin{\Box} R) &\equiv_M (P \sqcap Q) \mathbin{\Box} (P \sqcap R) \\
P \mathbin{\Box} \mathrm{STOP} &\equiv_M P \\
(a \rightarrow (P \sqcap Q)) &\equiv_M (a \rightarrow P) \sqcap (a \rightarrow Q) \\
(a \rightarrow P) \mathbin{\Box} (a \rightarrow Q) &\equiv_M (a \rightarrow P) \sqcap (a \rightarrow Q) \\
P \sqcap P &\equiv_M P \\
P \sqcap Q &\equiv_M Q \sqcap P \\
P \sqcap (Q \sqcap R) &\equiv_M (P \sqcap Q) \sqcap R \\
P \parallel Q &\equiv_M Q \parallel P \\
P \parallel (Q \parallel R) &\equiv_M (P \parallel Q) \parallel R \\
P \parallel (Q \sqcap R) &\equiv_M (P \parallel Q) \sqcap (P \parallel R) \\
(a \rightarrow P) \parallel (b \rightarrow Q) &\equiv_M \mathrm{STOP} \qquad \text{if } a \neq b \\
&\equiv_M (a \rightarrow (P \parallel Q)) \quad \text{if } a = b \\
P \parallel \mathrm{STOP} &\equiv_M \mathrm{STOP} \\
P|||Q &\equiv_M Q|||P \\
(P|||Q)|||R &\equiv_M P|||(Q|||R) \\
P|||(Q \sqcap R) &\equiv_M (P|||Q) \sqcap (P|||R) \\
(a \rightarrow P)|||(b \rightarrow Q) &\equiv_M (a \rightarrow (P|||(b \rightarrow Q))) \mathbin{\Box} (b \rightarrow ((a \rightarrow P)|||Q)) \\
P; (Q; R) &\equiv_M (P; Q); R \\
\mathrm{STOP}|||Q &\equiv_M Q \\
\mathrm{SKIP}; Q &\equiv_M Q \\
\mathrm{STOP}; Q &\equiv_M \mathrm{STOP} \\
P; (Q \sqcap R) &\equiv_M (P; Q) \sqcap (P; R) \\
(P \sqcap Q); R &\equiv_M (P; R) \sqcap (Q; R) \\
(a \rightarrow P); Q &\equiv_M (a \rightarrow P; Q) \qquad \text{if } a \neq \surd \\
(P \backslash a) \backslash b &\equiv_M (P \backslash b) \backslash a \\
(P \backslash a) \backslash a &\equiv_M P \backslash a \\
(a \rightarrow P) \backslash b &\equiv_M (a \rightarrow P \backslash b) \qquad \text{if } a \neq b \\
&\equiv_M P \backslash b \qquad\qquad \text{if } a = b \\
(P \sqcap Q) \backslash a &\equiv_M (P \backslash a) \sqcap (Q \backslash a)
\end{aligned}
$$

TABLE 1

3. The new model.

The failures model is unable to provide an adequate treatment of the phenomenon of *divergence.* In addition, all refusal sets were taken to be finite. The new model has the same basic structure but with two modifications.

1. Divergence. A process is said to diverge at some stage if it is possible for it to engage in an unbounded sequence of internal actions, invisible to its environment. Such behaviour is introduced when hiding an infinite sequence of events; if such a sequence is rendered invisible to the environment of the process, the resulting process diverges. An example of this is provided by the term

$$(\mu p.(a \to p))\backslash a.$$

Here the recursively defined process is able to perform an unbounded sequence of a events, which become internal actions when the hiding operation is applied. Divergence is also introduced by ill-defined recursive definitions, because we regard the initiation of a recursive call as an internal action; an example is provided by the recursive term $\mu p.p$, whose execution results in an infinite sequence of recursive calls. In the failures model M a divergence caused by hiding was modelled as CHAOS, although another plausible version of the hiding operation regarded this type of divergence as indistinguishable from deadlock. Some of the properties of this version of hiding were investigated in [2]. This alternate form of hiding, which models divergence by STOP and thus identifies divergence with deadlock, does not have such appealing algebraic properties, and the chaotic form was preferred in [3,4]. In particular, the chaotic form of hiding is a continuous operation, unlike the deadlocking version. However, CHAOS is simply the process which can at any stage in its execution *refuse* any set of events; that is, CHAOS *always responds* to its environment by either refusing or performing an event. It can therefore be argued that it is unreasonable to identify divergence with CHAOS, the ability to *fail to respond* at all to the environment, since divergence is more accurately represented by the *inability* to respond in any finite time. Indeed, CHAOS does not possess all of the combinational properties we would like to associate with a diverging process. In particular, the following equivalences do not generally hold in the failures model:

$$\begin{aligned} P \,\square\, \text{CHAOS} &\equiv_M \text{CHAOS}, \\ P \parallel \text{CHAOS} &\equiv_M \text{CHAOS}, \\ P \,|||\, \text{CHAOS} &\equiv_M \text{CHAOS}, \\ \text{CHAOS}; Q &\equiv_M \text{CHAOS}. \end{aligned}$$

In each of these cases we would expect divergence of the component process to cause the possibility of divergence in the compound process. Similar problems were encountered by Hennessy and de Nicola [12,19], in trying to axiomatize the failures model, and by Roscoe [23] when trying to make connections between the failures semantics and an operational interpretation of process behaviour. The use of CHAOS for the purpose of modelling divergence does not quite fit properly with operational intuitions. Thus, the failures model alone is insufficiently powerful to give a satisfactory or convincing account of divergence. In order to provide a more pleasing treatment of divergence, we introduce an extra component into the semantic description of a process. In addition to a failure set, a process will be associated with a *divergence set*; this will be a set of traces. If s is a divergence trace of a process we interpret this as saying that the process may be diverging once it has performed the sequence s.

2. Infinite refusal sets. Secondly, if Σ is infinite we will allow refusal sets to be infinite; but we also add a closure condition which makes this change largely cosmetic. Specifically, an infinite set will be a possible refusal if and only if all of its finite subsets are refusable. Thus, infinite refusal sets are determined by their finite subsets being refusable.

In this new model, which we will denote N, processes are modelled as pairs

$$\langle F, D\rangle$$

with

$$F \subseteq \Sigma^* \times \mathcal{P}(\Sigma),$$
$$D \subseteq \Sigma^*.$$

In such a pair $\langle F, D\rangle$ the failure set is F and the divergence set is D. We will extract these two components with the functions *failures* and *div*. We require the following conditions on F, which should be compared with (M1)–(M4) of the previous section.

$$(\langle\rangle, \emptyset) \in F \tag{N1}$$
$$(st, \emptyset) \in F \Rightarrow (s, \emptyset) \in F \tag{N2}$$
$$(s, X) \in F \ \&\ Y \subseteq X \Rightarrow (s, Y) \in F \tag{N3}$$
$$(s, X) \in F \ \&\ (\forall c \in Y. ((s\langle c\rangle, \emptyset) \notin F)) \Rightarrow (s, X \cup Y) \in F \tag{N4}$$
$$(\forall Y \in p(X). (s, Y) \in F) \Rightarrow (s, X) \in F. \tag{N5}$$

The only difference between (N1)–(N4) and the previous conditions is that Y is allowed to be infinite in (N4), whereas from use of (M4) only finite sets of impossible events can be included. (N5) states that a set is refusable if all of its finite subsets are refusable; the converse is implied by (N3).

We also impose a condition on the divergence set, corresponding to the intuition that divergence is a persistent phenomenon: once a process is diverging it diverges forever. Moreover, it is impossible to determine finitely any information about a diverging process, so that we cannot rule out the possibility that it might engage at some stage in some sequence of events. In other words, we regard divergence as *catastrophic*. These considerations lead us to formulate some conditions relating the divergence set D and failure set F

of a process:

$$s \in D \Rightarrow st \in D \tag{D1}$$
$$s \in D \Rightarrow (st, X) \in F. \tag{D2}$$

Condition (D1) states that the divergence set of a process is *suffix-closed.* The other condition states the catastrophic or chaotic nature of divergence. A similar argument was used in [3,4] to suggest that the failure set of a diverging process should be the most nondeterministic.

As in the old model, there is a natural partial order on the set of pairs $\langle F, D\rangle$:

$$\langle F_1, D_1\rangle \sqsubseteq \langle F_2, D_2\rangle \Leftrightarrow F_1 \supseteq F_2 \ \& \ D_1 \supseteq D_2.$$

The interpretation of this is that a process P_1 is more nondeterministic than P_2 if it can diverge whenever P_2 can diverge and fail whenever P_2 can fail. Again this ordering produces a complete semi-lattice structure; the least element, the most nondeterministic process, denoted $\perp$, has divergence set Σ^* and failure set $\Sigma^* \times \mathcal{P}(\Sigma)$. Since our model identifies all terms which diverge, we find it convenient to abuse notation slightly and introduce a constant term $\perp$ to the syntax of TCSP, representing divergence explicitly.

We say that a process is *divergence-free* if its divergence set is empty. The divergence-free processes form a semi-lattice which is clearly isomorphic to the old failures model, with bottom element CHAOS. The isomorphism $\Phi : M \to N$, given by

$$\Phi(F) = (\{(s, X) \mid \forall Y \in p(X).(s, Y) \in F\}, \emptyset),$$

merely assumes no divergence and introduces infinite refusal sets when required by (N5). Thus, in this model there is a distinction between the processes

$$\begin{aligned} \perp &= \langle \Sigma^* \times \mathcal{P}(\Sigma), \Sigma^*\rangle, \\ \Phi(\text{CHAOS}) &= \langle \Sigma^* \times \mathcal{P}(\Sigma), \emptyset\rangle. \end{aligned}$$

Note that the *deterministic* processes in the new model (the maximal elements) are precisely the images of deterministic processes in the old model, under this isomorphism. In particular, deterministic processes are divergence-free.

Semantics.

To give a semantics to our TCSP language we define a mapping $\mathcal{N}$ from processes to failure sets and divergence sets, when supplied with an environment for the meanings of free identifiers. Now we need an environment which maps process identifiers to pairs $\langle F, D\rangle$. Thus, an environment e will be a function $e : \text{TCSP} \to N$. As remarked earlier, we use the functions *failures* and *div* to extract the two components of a pair (F, D). We write $e + [p \mapsto (F, D)]$ for the environment which agrees with e except at p, which is mapped to the given pair. Let NEnv be the set of environments of this type. The type of the semantic function is thus

$$\mathcal{N} : \mathrm{TCSP} \to [\mathrm{NEnv} \to N].$$

For presentation purposes it is sometimes convenient to factor $\mathcal{N}$ into two component functions, by defining auxiliary semantic functions $\mathcal{F}$ and $\mathcal{D}$ such that

$$\mathcal{N}[\![P]\!]e = (\mathcal{F}[\![P]\!]e, \mathcal{D}[\![P]\!]e).$$

With this notation, $\mathcal{F}[\![P]\!]e$ is the failure set of P and $\mathcal{D}[\![P]\!]e$ is the divergence set. This enables us, when desirable, to define $\mathcal{N}[\![P]\!]$ in terms of the two components. Strictly speaking, the intention is to define both components simultaneously (using mutual recursion). This is illustrated in the definition for recursive terms:

$$\mathcal{N}[\![\mu p.P]\!]e = \mathrm{fix}(\lambda(F, D).\mathcal{N}[\![P]\!](e + [p \mapsto (F, D)])).$$

For the other syntactic constructs, we define the divergence semantics first and then give the failure sets. For the other syntactic constructs, the divergence semantics

$$\mathcal{D} : \mathrm{TCSP} \to [\mathrm{NEnv} \to \mathcal{P}(\Sigma^*)]$$

is provided by the clauses:

$$\begin{aligned}
\mathcal{D}[\![p]\!]e &= \mathrm{div}(e[\![p]\!]) \\
\mathcal{D}[\![\mathrm{STOP}]\!]e &= \emptyset \\
\mathcal{D}[\![\mathrm{SKIP}]\!]e &= \emptyset \\
\mathcal{D}[\![a \to P]\!]e &= \{\, \langle a \rangle s \mid s \in \mathcal{D}[\![P]\!]e \,\} \\
\mathcal{D}[\![P \sqcap Q]\!]e &= \mathcal{D}[\![P]\!]e \cup \mathcal{D}[\![Q]\!]e \\
\mathcal{D}[\![P \,\Box\, Q]\!]e &= \mathcal{D}[\![P]\!]e \cup \mathcal{D}[\![Q]\!]e \\
\mathcal{D}[\![P || Q]\!]e &= \{\, st \mid s \in (\mathcal{D}[\![P]\!]e \cap \mathrm{traces}(\mathcal{F}[\![Q]\!]e)) \cup (\mathcal{D}[\![Q]\!]e \cap \mathrm{traces}(\mathcal{F}[\![P]\!]e)) \,\} \\
\mathcal{D}[\![P ||| Q]\!]e &= \{\, u \mid \exists s, t.\, u \in \mathrm{merge}(s, t)\ \& \\
&\qquad (s \in \mathcal{D}[\![P]\!]e\ \&\ t \in \mathrm{traces}(\mathcal{F}[\![Q]\!]e)\ \vee\ t \in \mathcal{D}[\![Q]\!]e\ \&\ s \in \mathrm{traces}(\mathcal{F}[\![P]\!]e)) \,\} \\
\mathcal{D}[\![P;Q]\!]e &= \mathcal{D}[\![P]\!]e \cup \{\, st \mid s \text{ is tick-free }\ \&\ s\surd \in \mathrm{traces}(\mathcal{F}[\![P]\!]e)\ \&\ t \in \mathcal{D}[\![Q]\!]e \,\} \\
\mathcal{D}[\![P \backslash a]\!]e &= \{\, (s \backslash a)t \mid s \in \mathcal{D}[\![P]\!]e \,\} \cup \{\, (s \backslash a)t \mid \forall n. sa^n \in \mathrm{traces}(\mathcal{F}[\![P]\!]e) \,\} \\
\mathcal{D}[\![f^{-1}(P)]\!]e &= \{\, s \mid f(s) \in \mathcal{D}[\![P]\!]e \,\} \\
\mathcal{D}[\![f(P)]\!]e &= \{\, f(s)t \mid s \in \mathcal{D}[\![P]\!]e \,\}
\end{aligned}$$

Notice from this definition that STOP and SKIP have empty divergence sets, while a nondeterministic composition $P \sqcap Q$ or $P \,\Box\, Q$ may diverge if one of the components diverges. In a parallel composition $P||Q$ or $P|||Q$ divergence can start at some stage if either of the component processes can diverge. A sequential composition $P;Q$ can diverge if either the first component diverges or if the second diverges after the first one has terminated successfully. The hiding operation explicitly introduces divergence in a case where the original process is capable of unboundedly many hidden actions; this accords with our intuitions about divergence, as stated above. Finally, the divergent traces of a renamed process are obtained by renaming from those of the original process. It should be noted that each of these divergence set constructions preserves property (D1).

The failures semantic function has type:

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

We have already specified $\mathcal{F}[\![\mu p.P]\!]e$. For the other syntactic constructs, we specify the following clauses. Apart from the need to close up under condition (D2), these definitions are essentially those of [4], and the reader will find further explanation there.

$$\begin{aligned}
\mathcal{F}[\![p]\!]e &= \text{failures}(e[\![p]\!]) \\
\mathcal{F}[\![\text{STOP}]\!]e &= \{ (\langle\rangle, X) \mid X \in \mathcal{P}(\Sigma) \} \\
\mathcal{F}[\![\text{SKIP}]\!]e &= \{ (\langle\rangle, X) \mid \surd \notin X \} \cup \{ (\surd, X) \mid X \in \mathcal{P}(\Sigma) \} \\
\mathcal{F}[\![a \to P]\!]e &= \{ (\langle\rangle, X) \mid a \notin X \} \cup \{ (\langle a\rangle s, X) \mid (s, X) \in \mathcal{F}[\![P]\!]e \} \\
\mathcal{F}[\![P \sqcap Q]\!]e &= \mathcal{F}[\![P]\!]e \cup \mathcal{F}[\![Q]\!]e \\
\mathcal{F}[\![P \,\Box\, Q]\!]e &= \{ (\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}[\![P]\!]e \cap \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \neq \langle\rangle \;\&\; (s, X) \in \mathcal{F}[\![P]\!]e \cup \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P \,\Box\, Q]\!]e \} \\
\mathcal{F}[\![P||Q]\!]e &= \{ (s, X \cup Y) \mid (s, X) \in \mathcal{F}[\![P]\!]e \;\&\; (s, Y) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P||Q]\!]e \} \\
\mathcal{F}[\![P|||Q]\!]e &= \{ (u, X) \mid \exists s, t.\, (s, X) \in \mathcal{F}[\![P]\!]e \;\&\; (t, X) \in \mathcal{F}[\![Q]\!]e \;\&\; u \in \text{merge}(s, t) \} \\
&\quad \cup \{ (u, X) \mid u \in \mathcal{D}[\![P|||Q]\!]e \} \\
\mathcal{F}[\![P;Q]\!]e &= \{ (s, X) \mid s \text{ tick-free } \& \, (s, X \cup \{\surd\}) \in \mathcal{F}[\![P]\!]e \} \\
&\quad \cup \{ (st, X) \mid (s\surd, \emptyset) \in \mathcal{F}[\![P]\!]e \;\&\; s \text{ tick-free } \& \; (t, X) \in \mathcal{F}[\![Q]\!]e \} \\
&\quad \cup \{ (s, X) \mid s \in \mathcal{D}[\![P]\!]e \} \\
\mathcal{F}[\![P \backslash a]\!]e &= \{ (s \backslash a, X) \mid (s, X \cup \{a\}) \in \mathcal{F}[\![P]\!]e \} \\
&\quad \cup \{ (u, X) \mid u \in \mathcal{D}[\![P \backslash a]\!]e \} \\
\mathcal{F}[\![f^{-1}(P)]\!]e &= \{ (s, X) \mid (f(s), f(X)) \in \mathcal{F}[\![P]\!]e \} \\
\mathcal{F}[\![f(P)]\!]e &= \{ (f(s), X) \mid (s, f^{-1}(X)) \in \mathcal{F}[\![P]\!]e \} \cup \{ (s, X) \mid s \in \mathcal{D}[\![f(P)]\!]e \}.
\end{aligned}$$

The next result establishes that our semantic definitions make sense.

THEOREM 1. *All CSP operations defined above are well defined and continuous.*

Proof. Well definedness is easy to show, except for the synchronous parallel operator. In each case we have to establish that the failure set operations and divergence set operations corresponding to the syntactic constructions preserve the properties (N1)–(N5) and (D1)–(D2). Only the proof for the synchronous parallel operator is non-trivial. A full proof may be found in [23] or in the full version of this paper [5]. Continuity proofs are relatively straightforward, along the lines of the proofs given in [4],[2], and [23]. ∎

Since all of our operators are continuous, we can justify our use of least fixed points in defining the meaning of recursive definitions, and we know that these fixed points are explicitly constructible, as was the case in the earlier model.

Examples.

1. The process defined by the recursion $P = (a \to P)$ is denoted $\mu p.(a \to p)$. This process has:

$$\begin{aligned} \mathcal{F}[\![\mu p.(a \to p)]\!]e &= \{(a^n, X) \mid n \geq 0 \ \& \ a \not\in X\} \\ \mathcal{D}[\![\mu p.(a \to p)]\!]e &= \emptyset. \end{aligned}$$

2. The recursive term $\mu p.p$ denotes the most nondeterministic process $\perp$, which can do anything at all:

$$\begin{aligned} \mathcal{F}[\![\mu p.p]\!]e &= \Sigma^* \times \mathcal{P}(\Sigma), \\ \mathcal{D}[\![\mu p.p]\!]e &= \Sigma^*. \end{aligned}$$

Note that our notation implies that

$$\begin{aligned} \mathcal{M}[\![\mu p.p]\!]\rho &= \text{CHAOS} \\ \mathcal{N}[\![\mu p.p]\!]e &= \perp. \end{aligned}$$

This is an example in which the two semantics produce distinct results. We should be careful to distinguish between the meanings of terms in the two models. However, we can show that the old semantics and the new essentially coincide except in their treatment of divergence. This is stated precisely as follows. First we need to define an appropriate notion of matching between the environments used in the $\mathcal{M}$ semantics and those used in the $\mathcal{N}$ semantics.

Definition. The operation $\Phi : M \to N$ induces a function $\Phi : R \to \text{Env}$ by:

$$(\Phi\rho)[\![p]\!] = \langle \Phi(\rho[\![p]\!]), \emptyset \rangle.$$

The environments ρ and $\Phi\rho$ can be said to *match.*

LEMMA 1. *If P is a TCSP term, then for all ρ,*

$$\mathcal{N}[\![P]\!](\Phi\rho) \sqsubseteq_N \Phi(\mathcal{M}[\![P]\!]\rho).$$

THEOREM 2. *If P is a TCSP term and ρ an environment such that $\mathcal{D}[\![P]\!](\Phi\rho) = \emptyset$, then*

$$\mathcal{N}[\![P]\!](\Phi\rho) = \Phi(\mathcal{M}[\![P]\!]\rho).$$

Many algebraic properties of processes and these operators can be proved. The identities listed in Table 1 for the failures model are also true in this model (with $\equiv_M$ replaced by $\equiv_N$), except that

$$P \,\|\, \mathrm{STOP} \equiv_N \mathrm{STOP}$$

is valid only if $P \not\equiv_N \bot$, as we regard divergence as catastrophic. It is important to remember, then, that not all equivalences remain true in the passage from the failures model to the extended model, because of the superior treatment of divergence in the new model. Additional identities for the new semantics include the following.

$$\begin{aligned} P \,\Box\, \bot &\equiv_N \bot \\ P \sqcap \bot &\equiv_N \bot \\ P \,\|\, \bot &\equiv_N \bot \\ \bot; Q &\equiv_N \bot \\ \bot \backslash a &\equiv_N \bot \end{aligned}$$

TABLE 2

The fixed point theorem also holds in the new model. We have the identity

$$\mu p.P \equiv_N [(\mu p.P)\backslash p]P.$$

Strictness.

It can be argued [4] that most of the operators introduced so far ought to be *strict*, in that they should preserve divergence. The identities listed in Table 2 reflect this property. An exception is the prefixing operation $(a \rightarrow P)$, where divergence of P cannot manifest itself until after the initial occurrence of a; a similar exception is the second argument of a sequential composition, whose divergence cannot come into effect until the first component has terminated. Some further exceptions to strictness will be discussed in the next section, where we define the semantics of some operations introduced in [3,4].

4. Further operators.

It is possible to devise many useful operations, notably some interesting forms of parallel composition. In this section we revise the definitions of a few interesting forms of composition which were described in [3,4], bringing out certain inadequacies in the earlier treatment of divergence and showing that a cleaner treatment is obtained with our new definitions. By redefining the semantics of these operations in this way we achieve a better match with operational intuitions.

1. Mixed parallel composition.

We can define a parallel composition in which two processes operate with named alphabets and are required to cooperate on events common to both of their alphabets, but may progress independently on events belonging solely to their own alphabet. This mixed parallel composition is less restrictive than the *synchronous* version and not as generous as the *asynchronous* form. It is closely related to the *ignoring* operator of [3,4] and to the mixed parallel composition of [2,23]. When P and Q are to run in parallel, with P using alphabet A and Q using alphabet B, the resulting process is denoted:

$$[P_A \|_B Q].$$

Its divergence set and failure set, built up as usual from those of the constituent processes, are:

$$\begin{aligned}\mathcal{D}[\![P_A \|_B Q]\!]e &= \{\, uv \mid u \in (A \cup B)^* \ \&\ \text{either } ((u{\restriction}A \in \mathcal{D}[\![P]\!]e \ \&\ u{\restriction}B \in \text{traces}(\mathcal{F}[\![Q]\!]e)) \\ &\qquad\qquad \text{or } (u{\restriction}A \in \text{traces}(\mathcal{F}[\![P]\!]e) \ \&\ u{\restriction}B \in \mathcal{D}[\![Q]\!]e))\,\} \\ \mathcal{F}[\![P_A \|_B Q]\!]e &= \{\, (u, X \cup Y \cup Z) \mid u \in (A \cup B)^*, X \subseteq A, Y \subseteq B, Z \subseteq \overline{A \cup B}, \\ &\qquad (u{\restriction}A, X) \in \mathcal{F}[\![P]\!]e, (u{\restriction}B, Y) \in \mathcal{F}[\![Q]\!]e \,\} \cup \{\, (u, X) \mid u \in \mathcal{D}[\![P_A \|_B Q]\!]e \,\}.\end{aligned}$$

Here we introduce the notation $u{\restriction}A$ for the trace resulting from u after the removal of all events outside of the set A. We also use $\overline{C}$ for the complement of a set C. According to this definition, the traces u of $P_A \|_B Q$ are built up from events in A and B, and filtering out only those events which belong to A produces a trace $u{\restriction}A$ of P, while filtering out the events in B produces a trace $u{\restriction}B$ of Q. The compound process diverges after performing the sequence u if either P can diverge after $u{\restriction}A$ or Q can diverge after $u{\restriction}B$. Events in A are refused if P refuses them, while Q chooses whether or not to perform the events in B. Events common to A and B can be refused by either process. Events outside of A and B are always impossible.

It is easy to check, given the well-definedness of the synchronous parallel composition [5], that this construction produces a process when applied to processes, *i.e.* that conditions (N1)–(N5) and (D1)–(D2) are preserved. The following *associativity* property can also be proved; see [2] for details.

LEMMA 3. *For all processes P, Q, R, and all sets of events A, B, C,*

$$[P_A \|_{B\cup C} [Q_B \|_C R]] \equiv_N [[P_A \|_B Q]_{A\cup B} \|_C R].$$

In view of this result, this notation generalizes to a parallel composition of more than three processes. Given an indexed collection $V = \{ (P_i, A_i) \mid 1 \leq i \leq n \}$ we will write

$$\mathrm{PAR}(V) = \|_{i=1}^{n} (P_i, A_i)$$

for the parallel composition. Using this notation the mixed parallel composition $[P_A \|_B Q]$ may be rendered $(P, A) \| (Q, B)$. This type of composition can be useful in analysing the deadlock behaviour of networks of parallel processes, as shown in [6].

Another interesting identity concerns the result of hiding an event which is involved in a parallel composition. This identity did not in general hold in the old model, in some cases where hiding the event introduces divergence.

LEMMA 4. *Let A, B be subsets of Σ, let $c \notin B$, and let $C = A \cup \{c\}$. Then for all processes P, Q,*

$$[P_C \|_B Q] \backslash c \equiv_N [(P\backslash c)_A \|_B Q].$$

This result is important in analysing the effect of hiding internal communications in networks of communicating processes. It enables us to move hiding operators (in some cases) inside a parallel composition. This result is used in [6] to prove some useful results on deadlock analysis.

If we wish to run P and Q in parallel using alphabets A and B, we use the composition $[P_A \|_B Q]$ as above; events in the intersection $A \cap B$ are synchronized and correspond to communications between the two processes. These internal events may be concealed from the environment by applying the hiding operation. Provided this intersection is finite, we can define the process in which these internal communications are hidden as:

$$[P_A \|_B Q] \backslash (A \cap B).$$

This makes sense because hiding is associative. It is convenient to introduce a notation for this composition: we will denote it $[P\ {}_A \Leftrightarrow {}_B Q]$. Now if we wish to extend this to a network of several processes we may do so. The key associativity property is as follows. Provided $A \cap B \cap C = \emptyset$, and provided each of $A \cap B$, $B \cap C$, and $C \cap A$ is finite, we have:

$$[[P\ {}_A \Leftrightarrow {}_B Q]\ {}_{(A\cup B)} \Leftrightarrow {}_C R] \equiv_N [P\ {}_A \Leftrightarrow {}_{(B\cup C)} [Q\ {}_B \Leftrightarrow {}_C R]].$$

This follows from Lemmas 3 and 4.

2. Chaining.

In [3,4] we also defined a form of "chaining", a parallel composition $P \gg Q$ in which all outputs of P are fed into Q as inputs and hidden from their common environment. Assume that all events are communications between processes along named channels. An event consists of two parts m.t, where m is a channel name and t a value. Normally, the channel name "in" is associated with input, and "out" with output. We use the abbreviations

$$\begin{array}{rcl} (?x{:}T \rightarrow P(x)) & \text{for} & \Box_{t\in T}\,(\text{in}.t \rightarrow P(t)), \\ !t & \text{for} & (\text{out}.t \rightarrow \text{SKIP}). \end{array}$$

It is convenient also to allow the abbreviated form $?t$ to stand for the correspoding event. In order to cope with a form of channel naming, we also use the abbreviations:

$$\begin{array}{rcl} (\text{c}?x{:}T \rightarrow P(x)) & \text{for} & \Box_{t\in T}\,(\text{c.in}.t \rightarrow P(t)), \\ c!t & \text{for} & (\text{c.out}.t \rightarrow \text{SKIP}). \end{array}$$

We also allow the abbreviated form $c?t$ to stand for the corresponding event. The construct $(c?x{:}T \rightarrow P(x))$ represents a process which initially *inputs* on channel c a value for x from the set T (a value of type T); similarly, $c!t$ represents output of the value t along the channel. For example, a simple buffer of type T using input channel *in* and output channel *out* is:

$$B_1 = \mu p.(\text{in}?x{:}T \rightarrow \text{out}!x; p).$$

The chaining operation can be defined by combining a renaming with the mixed parallel operation and hiding. First we rename the output events of P and the input events of Q so that they become identical events; then we run the renamed versions of P and Q in parallel, using the renamed alphabets. This forces P to synchronize its outputs with the inputs of Q. Finally we hide all events common to these alphabets, which are precisely the internal communications between the two processes. Let α be a label distinct from *in* and *out*. Let swap(α, β) be the alphabet transformation defined:

$$\begin{array}{rcll} \text{swap}(\alpha,\beta)(x) & = & x & \text{if } x \notin \alpha.T \cup \beta.T \\ & = & \beta.t & \text{if } x = \alpha.t \quad (t \in T) \\ & = & \alpha.t & \text{if } x = \beta.t \quad (t \in T). \end{array}$$

Then if we put $A = \text{in}.T \cup \alpha.T$ and $B = \text{out}.T \cup \alpha.T$, we can define

$$(P \gg Q) = [\text{swap}(\text{out},\alpha)(P)\ {}_A \Leftrightarrow {}_B\ \text{swap}(\text{in},\alpha)(Q)].$$

As an example of the use of the chaining operation, the result of chaining two simple buffers B_1 together is again a buffer process, $B_1 \gg B_1$, capable of holding at most two values. Several interesting properties of buffer processes built from the chaining operation were discussed in [4]. Most of these carry over without problems to the new model. In particular, we have the identities $(B_n \gg B_m) = B_{n+m}$ for all $n, m \geq 0$.

Interestingly, the version of the chaining operation defined in the failures model can fail to be associative: it is not always true that $P \gg (Q \gg R)$ and $(P \gg Q) \gg R$ denote the same failure set. This property can fail to hold when divergence can arise between two of the processes, so that either $(P \gg Q)$ or $(Q \gg R)$ diverges. Recall that in the failures model divergence is represented by CHAOS; it is not generally true that

$$\text{CHAOS} \gg Q \equiv_M P \gg \text{CHAOS} \equiv_M \text{CHAOS}.$$

Again, the use of CHAOS to represent divergence is unsatisfactory. In the new model the chaining operation is associative. The identities

$$(\perp \gg Q) \equiv_N (P \gg \perp) \equiv_N \perp$$

are true for all P and Q.

LEMMA 5. *For all processes P, Q, R,*

$$P \gg (Q \gg R) \equiv_N (P \gg Q) \gg R.$$

9. Master-slave operation.

In [3,4] we also defined a "master-slave" construction $[P \parallel \text{m}{:}Q]$, in which the master process P refers to its slave Q by the name m. The communications between master and slave are hidden in this construction. The definition here is similar to the previous version, except that we do not make the construction strict in the "slave" argument. The reason for this is that we do not want a master-slave pair to diverge unless either the master is diverging or the slave has been asked to perform some action and is diverging. In other words, the master's activity will only be affected by a divergence of the slave if the master is actually waiting for a response from the slave. Let $C = T \cup \text{in}.T \cup \text{out}.T$ and let m be chosen to be distinct from *in* and *out*. We define first, for traces u, v a compatibility condition:

$$\text{compat}_{\text{m}}(u,v) \Leftrightarrow v \in C^* \;\&\; u\restriction(\text{m}.C) = \text{m.swap(in,out)}(v).$$

If u is a trace of the master process, then $u\restriction(\text{m}.C)$ is the sequence of events involving the slave named m. If we interchange the roles of input and output in this sequence we should get a trace of the slave process; this is the essence of the compatibility condition. For two compatible traces u and v, define $[u \parallel \text{m}{:}v]$ to be the trace $u\restriction(\Sigma - \text{m}.C)$. This is the sequence of events performed by the master process which do not involve the slave. Then we can define the master-slave operation on processes as follows:

$$\begin{aligned}
\mathcal{D}[\![P \parallel \text{m}{:}Q]\!]e = {} & \{\, [u \parallel \text{m}{:}v]w \mid \text{compat}_m(u,v) \;\&\; u \in \mathcal{D}[\![P]\!]e \;\&\; v \in \text{traces}(\mathcal{F}[\![Q]\!]e) \,\} \\
& \cup \{\, [u \parallel \text{m}{:}v]w \mid \text{compat}_m(u,v) \;\&\; u \in \text{traces}(\mathcal{F}[\![P]\!]e) \;\&\; v \in \mathcal{D}[\![Q]\!]e \;\&\; v \neq \langle\rangle \,\} \\
& \cup \{\, sw \mid \exists^{\infty}(u,v).\, u \in \text{traces}(\mathcal{F}[\![P]\!]e) \;\&\; v \in \text{traces}(\mathcal{F}[\![Q]\!]e) \;\&\; \text{compat}_m(u,v) \;\& \\
& \qquad s = [u \parallel \text{m}{:}v] \,\}.
\end{aligned}$$

For the failure set we define:

$$\begin{aligned}\mathcal{F}[\![P \parallel \mathrm{m}{:}Q]\!]e = &\ \{(s,X) \mid s \in \mathcal{D}[\![P \parallel \mathrm{m}{:}Q]\!]e\}\\ &\cup \{([u \parallel \mathrm{m}{:}v], X) \mid (u,U) \in \mathcal{F}[\![P]\!]e \ \&\ (v,V) \in \mathcal{F}[\![Q]\!]e \ \&\\ &\quad \mathrm{compat}_m(u,v) \ \&\ U \cup \mathrm{m.swap(in, out)}(V \cap C) = X \cup \mathrm{m}.C\}.\end{aligned}$$

Thus, the traces of the compound process are built from a master trace and a compatible slave trace; and an event not involving the slave process can be refused if the master process refuses it and there is no possibility of an internal action. On the other hand, an event involving the slave process may be refused if either the slave or the master refuses it.

The version of master-slave operation used in the old failures model had some slight problems, in particular in its behaviour in recursive definitions. For example, in the failures model we have the identity

$$\mu p.[(?x \rightarrow m!x) \parallel m{:}p] \equiv_M (?x \rightarrow \mathrm{STOP}),$$

although intuitively we can see that there is a possibility of divergence after the first input. In the new model, with the above definition, we do indeed get

$$\mu p.[(?x \rightarrow m!x) \parallel m{:}p] \equiv_N (?x \rightarrow \bot).$$

Another problem was that in the failures model, the order in which a master process binds his slaves could make a difference in the behaviour of the system. In other words, the following identity does not always hold in the failures model:

$$[[P \parallel m{:}Q] \parallel n{:}R] \equiv_M [[P \parallel n{:}R] \parallel m{:}Q] \quad (n \neq m).$$

Again this deficiency appears when divergence can occur between the master and one of the slaves. In the new model the order in which a master process binds his slaves is irrelevant, and we do have

$$[[P \parallel m{:}Q] \parallel n{:}R] \equiv_N [[P \parallel n{:}R] \parallel m{:}Q] \quad (n \neq m).$$

5. Conclusions.

The revised model of processes described here enjoys the mathematical properties of a complete semi-lattice under the componentwise ordering $\sqsubseteq_N$ introduced earlier. All of the techniques used in [4] to specify and prove properties of processes may be adapted with ease to this setting. In particular, it is possible to use the notions of *constructivity* and *non-destructivity* in the analysis of recursively defined processes. Thus, with minor modifications to fit with the revised definitions of some of the operators, the examples described in [4] and the proofs of their properties described in that paper can be reformulated in the revised setting.

The failures model of communicating processes was introduced in [3,4]. This model was itself an extension of Hoare's earlier *traces* model of processes [13], which was incapable of supporting any reasonable treatment of deadlock properties since it is impossible to represent the ability to refuse to perform an action in a model based solely on sequences of possible actions. Our motives in designing the failures model were therefore driven by a desire to model deadlock satisfactorily. Several other authors have also discovered models which can be related to failures. Milner's CCS [18] is founded on a rather different (more discriminating) notion of *observation equivalence*, and his synchronization trees provide an alternative framework in which our results can be formulated [2]. Our development of the failures model has clearly been strongly influenced by the work of Milner and his colleagues.

As we observed earlier, a model based on failures alone is inadequate for reasoning about the phenomenon of divergence. Problems related to this fact have been pointed out by [23], [12]. This led to the inclusion in [2] and [23] of an explicit and distinguished representation of divergence in the semantics of processes, producing models isomorphic to the one used in this paper. In a similar vein, Hennessy and de Nicola have constructed several models based on synchronization trees augmented by acceptance sets, and they have introduced the notion of *representation trees* [11,12,19,20]. Hennessy has pointed out in [11] that the model known as RT_2 is closely related to a submodel of ours based on extra assumptions on finite branching, although there are subtle differences between the treatments of internal actions in their model and ours. In fact, this submodel with finite branching can be thought of as containing all of the *denotable* elements of our model. Similar observations were made by de Nicola in [19,20], where he suggested an adjustment to the failures model to handle divergence in a more subtle way than was done in that model: this was the *Bounded Refusal Sets* model, and again this model can be seen as an alternative presentation of a submodel of ours. The full model N, as it stands, does allow a (pessimistic) treatment of unbounded nondeterminism, in the sense that many unboundedly nondeterministic processes can be represented in this model but any process will be identified with its closure.

If one focusses solely on the finitely branching submodel of ours, it is largely a matter of taste as to which presentation one prefers, as any theorem provable in one formulation of the model will be adaptable to the alternative settings. This is an observation due to Matthew Hennessy. He has proved in [11] some general results on the congruence of denotational and operational semantics and these can be adapted to our setting to demonstrate that our semantics is indeed in accordance with operational intuitions. It is possible to define an operational semantics for our language based on Milner's *synchronization trees*, extending the definitions of [2,23] and following the lines of the presentations in

[11,12,20,23]. Essentially the idea is that a term denotes a synchronization tree whose arcs are labelled by events or by a special symbol τ denoting an internal event. Recursively defined terms will in general denote infinite trees, and divergence corresponds to the presence of an infinite path of τ arcs. Each syntactic construct of our language then corresponds to an operation on synchronization trees. There is a natural notion of *implementability* of operations on these trees, and all of the CSP operations turn out to be implementable. Moreover, the denotational semantics of this paper can be shown to agree with the operational semantics, a property that failed for the earlier model because of its inadequate treatment of divergence.

Kennaway [15,16] described a model for processes from which failures can be derived [2], but in which the underlying partial ordering is different because of his decision to regard deadlock as disastrous. The notion of *implementation sets,* given in [2,4], is closely related to Kennaway's idea of a nondeterministic process as a set of deterministic processes. We defined a notion of *implementation* for the failures model of CSP. A deterministic process Q is said to *implement* a process P if $P \sqsubseteq_M Q$. For divergence-free processes in the new model the same ideas can be adapted. A divergence-free process can be identified with its set of deterministic implementations,

$$\mathrm{imp}(P) = \{\, Q \mid P \sqsubseteq_N Q \ \& \ Q \,\text{deterministic}\,\}.$$

In the absence of divergence, the CSP operations on processes are fully determined by their effect on deterministic processes. Thus, if $\circ$ is a binary CSP operation (such as $\|$), and if P, Q are divergence-free processes, we have

$$\begin{aligned}\mathcal{F}[\![P \circ Q]\!]e &= \bigcup\{\, \mathcal{F}[\![P' \circ Q']\!]e \mid P' \in \mathrm{imp}(P), Q' \in \mathrm{imp}(Q)\,\}\\ \mathcal{D}[\![P \circ Q]\!]e &= \bigcup\{\, \mathcal{D}[\![P' \circ Q']\!]e \mid P' \in \mathrm{imp}(P), Q' \in \mathrm{imp}(Q)\,\}.\end{aligned}$$

Similar results hold for unary CSP operations. As stated here, these definitions and results apply only to divergence-free processes. It is possible to extend these results to cover all processes, by redefining the notion of an implementation to include only "minimal" divergent processes. We will not discuss this issue here.

Darondeau [8] gave an "enlarged definition of observation congruence" for finite processes which essentially coincides with the failure equivalence [2] induced by focussing on failures; Darondeau's paper only considered finite terms. In [25] a model including acceptance sets in addition to refusal sets was discussed, although this seems not to possess such elegant mathematical properties as the failures model and appears as a result to be less well suited to analysis of deadlock. Olderog [21] introduced a model involving "readiness sets", which are analogous to acceptance sets; again there are connections with failure sets, since a readiness set can be regarded as the dual of a refusal set. The readiness model is based on a slightly different notion of equivalence than the one induced by failures. Broy introduces in [7] a somewhat complicated model in which rather more distinctions between processes are possible; in particular, he chooses not to regard the possibility of divergence as necessarily catastrophic (so that, for instance, $P \Box \bot \neq \bot$ in general in his model). His fixed point theory and operator definitions are made more complicated by this and by the intricate structure of his model.

There are several directions in which we want to develop our techniques and results. In the full version of this paper, proofs are given of some of the most interesting theorems. In [6] we discuss some useful results pertaining to the analysis of deadlock behaviour in networks of communicating processes. It is possible to adapt our semantics to imperative communicating processes such as Hoare's original CSP, in which processes have disjoint local states and can perform assignments to update their own state. An example in this vein is provided by Roscoe's semantics for *occam* [24], which arises from a failure set semantics by adjoining local states and building a "hybrid" semantic model. We plan to adopt similar techniques for CSP in a future paper. This should lead to a semantic model closely related to the linear history model developed by Pnueli, Lehmann and Francez [9], which uses *expectation sets* rather than refusal sets and is based on a different notion of equivalence. Connections with earlier models such as the one described in [10] should also become apparent. We also believe that this should lead us to a semantics supporting a partial correctness analysis which takes deadlock fully and explicitly into account, unlike many existing CSP semantics which have served as the basis for partial correctness reasoning. We hope to be able to make some connections with existing proof systems for CSP, such as those described in [1,17], and with Plotkin's structural operational semantics for CSP [22].

Acknowledgements.

The authors would like to thank C. A. R. Hoare for his many helpful suggestions and discussions, and for his encouragement and guidance during the development of this work. We have been strongly influenced by the work of Robin Milner, Matthew Hennessy and Rocco de Nicola. Discussions with them and with Bill Rounds and Ernst-Rudiger Olderog have been very useful.

6. References.

[1] Apt, K. R., Francez, N., and de Roever, W. P., A Proof System for Communicating Sequential Processes, ACM TOPLAS, Vol. 2 No. 3, July 1980.

[2] Brookes, S. D., A Model for Communicating Sequential Processes, Ph. D. thesis, Oxford University (1983). Available as CMU Technical Report CMU-CS-83-149 and PRG Monograph.

[3] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W., A Theory of Communicating Sequential Processes, Oxford University Computing Laboratory, Programming Research Group, Technical Report PRG-16.

[4] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W., A Theory of Communicating Sequential Processes, JACM July 1984.

[5] Brookes, S. D., and Roscoe, A. W., An Improved failures Model for Communicating Processes (full version of this paper), to appear, CMU Technical Report.

[6] Brookes, S. D., and Roscoe, A. W., Deadlock Analysis in Networks of Processes, to appear in Proceedings of the NATO Advanced Seminar on Concurrency, La Colle-Sur-Loup, Springer Verlag LNCS (1985).

[7] Broy, M., Semantics of Communicating Processes, preprint, Institut fur Informatik, Technische Universitat Munchen (1983).

[8] Darondeau, Ph., An enlarged definition and complete axiomatization of observational congruence of finite processes, Springer Verlag LNCS vol. 137, pp. 47-62 (1982).

[9] Francez, N., Lehmann, D., and Pnueli, A., A Linear History Semantics for Communicating Processes, Theoretical Computer Science 32 (1984) 25-46.

[10] Francez, N., Hoare, C. A. R., Lehmann, D., and de Roever, W. P., Semantics of nondeterminism, concurrency and communication, JCSS vol. 19 no. 3 (1979).

[11] Hennessy, M., Synchronous and Asynchronous Experiments on Processes, Information and Control, Vol. 59, Nos 1-3, pp. 36-83 (1983).

[12] Hennessy, M., and de Nicola, R., Testing equivalences for processes, Proc. ICALP 1983, Springer LNCS 154 (1983).

[13] Hoare, C. A. R., A Model for Communicating Sequential Processes, Oxford University Computing Laboratory, Programming Research Group, Technical Report PRG-22.

[14] Hoare, C. A. R., Communicating Sequential Processes, CACM 1978.

[15] Kennaway, J., Formal semantics of nondeterminism and parallelism, D. Phil thesis, Oxford University (1981).

[16] Kennaway, J., A theory of nondeterminism, Springer LNCS vol. 85, pp 338-350 (1980).

[17] Levin, G. M., and Gries, D., A Proof Technique for Communicating Sequential Processes, Acta Informatica 15 (1981).

[18] Milner, R., A Calculus of Communicating Systems, Springer Verlag LNCS 92.

[19] de Nicola, R., Two Complete Sets of Axioms for a Theory of Communicating Sequential Processes, Proc. International Conference on Foundations of Computation Theory, Borgholm, Sweden, Springer LNCS (1983).

[20] de Nicola, R., Models and Operators for Nondeterministic Processes, Proceedings of the Conference on Mathematical Foundations of Computer Science, Springer Verlag LNCS (1984).

[21] Olderog, E-R, Specification-oriented semantics of communicating processes, Proc. ICALP 1983, Springer LNCS 154 (1983).

[22] Plotkin, An Operational Semantics for CSP, W.G.2.2 Conference proceedings (1982).

[23] Roscoe, A. W., A Mathematical Theory of Communicating Processes, Ph. D. thesis, Oxford University (1982).

[24] Roscoe, A. W., A Denotational Semantics for *occam*, Proc. NSF-SERC Seminar on Concurrency, to appear in Springer Lecture Notes series (1984).

[25] Rounds, W. C., and Brookes, S. D., Possible futures, acceptances, refusals and communicating processes, Proc. 22nd IEEE Symposium on Foundations of Computer Science (1981).

[26] Stoy, J. E., Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory, MIT Press, Cambridge, Mass. (1977).