Models for Data-Flow Sequential Processes

Mark B. Josephs

Centre for Concurrent Systems and Very-Large-Scale Integration, Faculty of BCIM,
London South Bank University, 103 Borough Road, London SE1 0AA, UK
`josephmb@lsbu.ac.uk`

Abstract. A family of mathematical models of nondeterministic data flow is introduced. These models are constructed out of sets of traces, successes, failures and divergences, cf. Hoare's traces model, Roscoe's stable-failures model and Brookes and Roscoe's failures/divergences model of Communicating Sequential Processes. As in CSP, operators are defined that are convenient for constructing processes in the various models.

1 Introduction

Consider sequential processes that communicate via input streams and output streams (FIFO buffers of unlimited storage capacity), as in Kahn-MacQueen data-flow networks [17, 18]. They are capable of the following actions:

- selectively reading data from their input streams,
- unreading (pushing back) data to their input streams,
- writing data to their output streams, and
- termination.

Processes can be composed in parallel. In particular, an output stream of one process may be connected to an input stream of a second process. Any datum written to the output stream by the first process should be transferred (eventually and automatically) to the input stream, where it becomes available for reading by the second process.

Processes can also be composed in sequence. When one process terminates its successor starts to execute. An important point here (implicit in [10]) is that *termination does not destroy the contents of input streams and output streams.*

Some years ago, the author, Hoare and He [16, 9] devised a process algebra for (nondeterministic) data flow, as a variant of Communicating Sequential Processes (CSP) [12]. (Part of this work was reproduced in [13].) We showed how to simplify the failures/divergences model [4] of CSP so that refusal sets were no longer required; failures could instead be identified with (finite) 'quiescent traces' [7, 14] or 'traces of completed computation sequences' [23].

At the time we did not consider a binary angelic choice operator, nor sequential composition. One purpose of this article is to rectify those omissions. Note that termination is modelled in CSP by a special symbol $\surd$ (success) [11], but that would not work for what we shall call Data-Flow Sequential Processes

(DFSP). The solution is to create a 'stub' (a sequence of unread inputs) when termination occurs and there are no pending outputs, cf. [10, 6].

Another purpose of this article is to show how the more recent stable-failures model [27] of CSP can be adapted for DFSP. Indeed, a series of increasingly sophisticated models for DFSP will be introduced in a step-by-step manner, cf. [22]. Note that fairness issues, the focus of [23, 2, 3], are not addressed in these models.

The rest of this article is organised as follows. In Section 2, we recall the reordering relation [16, 9] between traces of directed events, a relation that captures the essence of data-flow communication. Subsequently, we define partial-correctness models (in Sections 3–5) and a total-correctness model (in Section 6) for DFSP, guided by what Roscoe [27] has done for CSP. In each case we consider the semantics of operators appropriate to the model. Conclusions are drawn in Section 7.

2 Directed Events, Traces and Reordering

A process is associated with an alphabet A, a (possibly infinite) set of symbols[1], partitioned into an input alphabet I and an output alphabet O. A symbol in I designates the transfer of a particular datum to a particular input stream; a symbol in O designates the transfer of a particular datum from a particular output stream. Such *directed events* are considered to be atomic, i.e., instantaneous.

Following Hoare [11], we define a *trace* to be a finite sequence (string) of symbols in A that expresses the occurrence of events over time as a linear order. In respect of a process that communicates through streams of unbounded capacity, however, two facts are noteworthy:

1. Events are independent if they are in the same direction but act upon different streams.
2. The occurrence of an input event does not depend upon the prior occurrence of an output event.

The first fact would justify taking a more abstract approach, namely, to follow Mazurkiewicz [21] by defining a trace to be an equivalence class on A^*. The two facts taken together would justify being more abstract still, namely, to follow Pratt [24] by defining a trace to be a partially-ordered multiset (pomset) on A. For example, if a and b are independent input events and c and d are independent output events, then the strings $cabd$ and $cbad$ are equivalent, but the only ordering between events is given by $a < d$ and $b < d$.

Anyway, the possibility of *reordering* a trace without affecting the behaviour of a process was recognized in [7, 28] and was formalised as a relation $t \ltimes u$ (t reorders u) between strings t and u in [16]. Reordering allows

[1] To be more concrete, we have in mind compound symbols with $s.d$ referring to stream s and datum d. We would then require that $s_0.d_0 \in I$ and $s_1.d_1 \in O$ implies that $s_0 \neq s_1$. Moreover, if D is a data type associated with stream s, then $s.d \in A$ for all $d \in D$.

– input symbols to be moved in front of other symbols
– output symbols to be moved behind other symbols

provided the symbols being swapped are associated with different streams. In other words, it is the strongest reflexive transitive relation (i.e. preorder) such that $tabu \ltimes tbau$ if $a \in I$ or $b \in O$, a and b designating transfers on different streams. For example, if a and b are independent input events and c and d are independent output events, then $badc \ltimes cabd$. Various properties of the reordering relation have been proved in [19, 20].

More abstractly, t and u are equivalent ($t \bowtie u$) if and only if $t \ltimes u$ and $u \ltimes t$. Note that, if two traces are equivalent, then reordering one into the other involves only the swapping of input symbols and the swapping of output symbols, not the swapping of input symbols with output symbols.

Not only does $\ltimes$ give us Mazurkiewicz's equivalence classes, but it also becomes a partial order on them. Note that a trace t is minimal (up to equivalence) if and only if $t \in I^*O^*$. Kahn [17] modelled a class of data-flow networks by means of continuous functions from the histories of input streams to the histories of output streams. For that class, the minimal traces are all that are needed. Moreover, reordering of a trace corresponds to 'augmentation' [25] or 'subsumption' [8], a partial order on pomsets.

3 Traces Model

A process with alphabet A (partitioned into I and O) can be modelled by a set T of traces, i.e., $T \subseteq A^*$. (Pratt [24] similarly models a process by a set of pomsets, and Gischer [8] investigates closure under subsumption.) This model avoids the Brock-Ackerman anomaly [1]. It embodies the following assumptions:

1. Divergence is always possible, i.e., a process may remain unstable indefinitely.
2. Quiescence (also referred to as stable failure) is always possible, i.e., a process that has become stable may refuse to output.

3.1 Healthiness conditions

Four conditions must be satisfied by such a set T:
It contains the empty sequence.

$$\varepsilon \in T \tag{1}$$

It is prefix-closed.[2]

$$\{t, u : tu \in T : t\} \subseteq T \tag{2}$$

It is receptive.

$$TI^* \subseteq T \tag{3}$$

[2] The set comprehension $\{l : D : E\}$ denotes the set of all values E obtained by substituting values that satisfy domain predicate D for the variables in the list l.

It is closed under reordering.

$$\{t, u \,:\, t \in T \wedge u \ltimes t \,:\, u\} \subseteq T \tag{4}$$

Observe that the space of healthy sets of traces is a complete lattice, with least (greatest) member A^* and greatest (least) member I^*, under the superset (subset) order. Also, for any non-empty subset S of healthy sets of traces, $\bigcap S$ is the least upper (greatest lower) bound and $\bigcup S$ is the greatest lower (least upper) bound.

3.2 Operators

It is convenient to construct a process P out of CSP-like operators. traces(P) is then the set of traces denoted by P. For a given I and O, P is refined by Q if traces$(P) \supseteq$ traces(Q). It is essential that the operators preserve the healthiness conditions and are monotonic with respect to the refinement order. It turns out that the operators are also continuous in the reverse (subset) order [27].

Quiescence. The process **stop** does nothing, though data can always be transferred to its input streams. Thus

traces(**stop**) $= I^*$

and we can see that **stop** refines every other process in the traces model.

Recursion. The meaning of a recursively-defined process $\mu X.\, F(X)$ is given by $\bigcup_{0 \leq i} F^i(\mathbf{stop})$, the least fixed point of continuous function F with respect to the subset order.

Nondeterministic choice. The process $P_0 \sqcap P_1$ behaves like P_0 or like P_1. (Broy [5] calls this 'erratic' choice because it is outside the control of the environment.)

traces$(P_0 \sqcap P_1) =$ traces$(P_0) \cup$ traces(P_1),

the greatest lower bound $\sqcap$ with respect to the superset order.

Conditional choice. Given a Boolean expression B, the process $P_0 \lhd B \rhd P_1$ behaves like P_0 or like P_1, depending upon whether or not B is true.

Prefixing. Given a minimal trace t (i.e. $t \in I^*O^*$), the process $t \rightarrow P$ reads data from its input streams and writes data to its output streams in the order given by t, and then behaves like P. The representation of such a sequence of internal data transfers as a single step is a convenient abstraction.

$\mathrm{traces}(t \to P) = I^* \cup \{u, v, w \,:\, u \in \mathrm{traces}(P) \wedge vw \ltimes tu \,:\, v\}$.[3]

stop is a fixed-point of input-prefixing:

$$\boxed{t \to \mathbf{stop} = \mathbf{stop}\,,\ \text{if } t \in I^*\,.}$$

Guarded choice. Quiescence and prefixing generalise to guarded choice: the process $|_{0 \le i < n} t_i \to P_i$ is constructed from an indexed set t of minimal traces (guards) and an indexed set P of processes.

$$\begin{aligned}&\mathrm{traces}(|_{0 \le i < n} t_i \to P_i)\\ =\ & I^* \cup \{i, u, v, w \,:\, 0 \le i < n \wedge u \in \mathrm{traces}(P_i) \wedge vw \ltimes t_i u \,:\, v\}\,.\end{aligned}$$

Observe that *in the traces model* guarded choice is simply a nondeterministic choice between prefixed processes.

After. The behaviour of P after the occurrence of t is given by the process P/t, for any trace $t \in \mathrm{traces}(P)$. In particular, P/t is always meaningful for $t \in I^*$ and behaves like P with the contents of its input streams determined by t.

$\mathrm{traces}(P/t) = \{u : tu \in \mathrm{traces}(P) : u\}$.

We have the following cancellation law:

$$\boxed{(t \to P)/u = P\,,\ \text{if } t \bowtie u\,.}$$

Observe also that

$$\boxed{P/t \text{ is refined by } P/u\,,\ \text{if } t \ltimes u\,,}$$

which follows from the definitions of refinement and /, and the property of $\ltimes$ that $t_0 \ltimes t_1$ implies $t_0 u \ltimes t_1 u$.

Parallel composition. Parallel composition corresponds to blending [28] (in which internal communication is concealed). Let input alphabet I_i and output alphabet O_i of P_i partition A_i, for $i = 0, 1$, with $I_0 \cap I_1 = \emptyset$ and $O_0 \cap O_1 = \emptyset$. Then the process $P_0 \parallel P_1$ has input alphabet $(I_0 \cup I_1) \setminus C$ and output alphabet $(O_0 \cup O_1) \setminus C$, where $C = A_0 \cap A_1$, and

$$\begin{aligned}&\mathrm{traces}(P_0 \parallel P_1)\\ =\ & \{t : t \in (A_0 \cup A_1)^* \wedge t \upharpoonright A_0 \in \mathrm{traces}(P_0) \wedge t \upharpoonright A_1 \in \mathrm{traces}(P_1) : t \setminus C\}\,.\end{aligned}$$

(Closure under reordering is proved in [19].) Pratt proposes a similar operator on sets of pomsets in [24].

Output-prefixing distributes through parallel composition, as follows:

$$\boxed{\begin{array}{l} t \to (P_0 \parallel P_1) = (t_0 \to P_0) \parallel (t_1 \to P_1)\,,\\ \text{where } t \bowtie t_0 t_1, \text{ if } t_0 \in (O_0 \setminus I_1)^* \text{ and } t_1 \in (O_1 \setminus I_0)^*\,.\end{array}}$$

[3] When the input streams referenced in t are each associated with a data type of cardinality one, the union with I^* is redundant.

Trading is also allowed between output-prefixing and after-input:

$$\boxed{(t \;\rightarrow\; P_0) \parallel P_1 = P_0 \parallel (P_1/t)\,,\ \text{if}\ t \in (O_0 \cap I_1)^*\,.}$$

4 Traces/Successes Model

A problem with the traces model is that it says nothing about successful termination and so one cannot define sequential composition. This is remedied by modelling the set S of successes, where $S \subseteq A^* \times I^*$. Often we are only interested in the first component of each pair in S, the set of such traces being $\text{dom}(S)$.

The structure of S (viz. a set of pairs of traces) is new[4] and can be understood as follows. For any $t \in A^*$, $u \in O^*$ and $v \in I^*$, $(tu, v) \in S$ records the ability of the process, after engaging in t, to terminate with u determining the contents of its output streams (i.e. pending outputs) and v determining the contents of its input streams (i.e. unread inputs). Of course, this implies that if $u = u_0u_1$, then the process is also able to terminate after tu_0, leaving u_1 pending and v unread. The second component of a member of S might be referred to as a 'stub', being what is left over upon termination of a process.

Broy and Lengauer [6] provide another way to model terminating processes. They generalise from deterministic processes represented by functions from states to states, to nondeterministic processes represented by sets of such functions.

4.1 Healthiness conditions

Three conditions must be satisfied by a pair (T, S) in addition to the four stated above for T:

The first component of every success is a trace.

$$\text{dom}(S) \subseteq T \tag{5}$$

S is receptive.

$$\{t, u, v \,:\, (t, u) \in S \wedge v \in I^* \,:\, (tv, uv)\} \subseteq S \tag{6}$$

S is closed under reordering of both components.

$$\{t, u, v, w \,:\, (t, u) \in S \wedge v \ltimes t \wedge w \bowtie u \,:\, (v, w)\} \subseteq S \tag{7}$$

Note that, in spite of Condition (6), symbols in the stub v of a success (u, v) are not necessarily present in u. Such successes can arise from the application of the after operator and from unreading, as we are about to see.

The space of healthy pairs (T, S) is a complete lattice under the pair-wise superset (subset) order.

[4] Roscoe discusses at length in [27] how termination is modelled in CSP with $\surd$. He does mention as an alternative, however, that 'the termination traces (at least) would have to be included as a separate component'.

4.2 Operators

The refinement order remains superset (now in each component). The operators previously introduced for the traces model can be lifted to the traces/successes model by defining their successes, and several more operators are now meaningful.

$\text{successes}(\mathbf{stop}) = \emptyset\,.$

$\text{successes}(P_0 \sqcap P_1) = \text{successes}(P_0) \cup \text{successes}(P_1)\,.$

$\text{successes}(t \;\rightarrow\; P) = \{u, v, w \;:\; (u, v) \in \text{successes}(P) \wedge w \ltimes tu \;:\; (w, v)\}\,.$

$\text{successes}(|_{0 \leq i < n} t_i \;\rightarrow\; P_i)$
$= \;\{i, u, v, w \;:\; 0 \leq i < n \wedge (u, v) \in \text{successes}(P_i) \wedge w \ltimes t_i u \;:\; (w, v)\}\,.$

$\text{successes}(P/t) = \{u, v \;:\; (tu, v) \in \text{successes}(P) \;:\; (u, v)\}\,.$

Parallel composition requires distributed termination [27] (in which both components must terminate before the composite process can).

$$\begin{aligned}\text{successes}(P_0 \parallel P_1) = \{t, u \;:\; & t \in (A_0 \cup A_1)^* \wedge u \in (I_0 \cup I_1)^* \\ & \wedge\; (t \upharpoonright A_0, u \upharpoonright I_0) \in \text{successes}(P_0) \\ & \wedge\; (t \upharpoonright A_1, u \upharpoonright I_1) \in \text{successes}(P_1) \;:\; (t \setminus C, u \setminus C)\}\,.\end{aligned}$$

Termination. The process **skip** differs from **stop** in that it terminates successfully.

$\text{traces}(\mathbf{skip}) = I^*$

$\text{successes}(\mathbf{skip}) = \{u, v \;:\; u \in I^* \wedge u \bowtie v \;:\; (u, v)\}\,.$

Reaction. The process t, where t is a minimal trace, reads data from its input streams and writes data to its output streams in the order given by t, and then terminates successfully. (So ε is the same as **skip**.)

$\text{traces}(t) = I^* \cup \{u, v, w \;:\; u \in I^* \wedge vw \ltimes tu \;:\; v\}$

$\text{successes}(t) = \{u, v \;:\; u \in I^* \wedge v \ltimes tu \;:\; (v, u)\}\,.$

It is a special case of prefixing:

$\boxed{t = t \;\rightarrow\; \mathbf{skip}\,,\ \text{if } t \in I^* O^*\,.}$

Unreading. The process t^{-1}, $t \in I^*$, unreads (pushes back) data on its input streams before terminating successfully. (So ε^{-1} is also the same as **skip**.)

$\text{traces}(t^{-1}) = I^*$

$\text{successes}(t^{-1}) = \{u, v \;:\; u \in I^* \wedge tu \bowtie v \;:\; (u, v)\}\,.$

Sequential composition. The process $P_0; P_1$ behaves like P_0 until that terminates successfully, allowing P_1 to take over.

$$\begin{aligned}&\text{traces}(P_0; P_1)\\ &= \text{traces}(P_0)\\ &\quad \cup \{t_0, t_1, t, u, v : (t_0, v) \in \text{successes}(P_0) \wedge vt_1 \in \text{traces}(P_1) \wedge tu \ltimes t_0t_1 : t\}\end{aligned}$$

$$\begin{aligned}&\text{successes}(P_0; P_1)\\ &= \{t_0, t_1, t, v, w : (t_0, v) \in \text{successes}(P_0)\\ &\qquad\qquad \wedge (vt_1, w) \in \text{successes}(P_1) \wedge t \ltimes t_0t_1 : (t, w)\}\,.\end{aligned}$$

Compare the following law to how division by a non-zero number relates to multiplication by the reciprocal of that number:

$$\boxed{P/t = t^{-1}; P\,,\ \text{if } t \in I^*\,.}$$

Iteration ${}^*P$ is a special case of recursion [12]: $\mu\, X.\, P; X$.

5 Traces/Successes/Failures Model

A problem with the above models is that quiescence is always a possibility. This is remedied by modelling the set F of (stable) failures, where $F \subseteq A^*$.

The structure of F (viz. a set of traces) is the same as in the author's earlier work [16, 15]. For any $t \in A^*$, $t \in F$ records the ability of the process, after engaging in t, to refuse to output after becoming stable, cf. Roscoe's stable-failures model[5] [27].

5.1 Healthiness conditions

Two conditions must be satisfied by a triple (T, S, F) in addition to the seven stated above for a pair (T, S):

F is a subset of T.

$$F \subseteq T \tag{8}$$

F is closed under reordering.

$$\{t, u : t \in F \wedge u \ltimes t : u\} \subseteq F \tag{9}$$

Note that whether or not a process, after engaging in $t \in T$, is able to terminate is independent of whether or not it is able to become quiescent. That is, $t \notin \text{dom}(S) \cup F$, $t \in \text{dom}(S) \setminus F$, $t \in F \setminus \text{dom}(S)$ and $t \in \text{dom}(S) \cap F$ are all possible.

The space of healthy triples (T, S, F) is a complete lattice under the component-wise superset (subset) order.

[5] Consider a trace t that can be extended by $u \in O^*$ to $tu \in F$ and, for simplicity, define a refusal set to be a set of streams. Then we may associate with t any refusal set consisting of output streams that are not involved in the events recorded in u [9].

5.2 Operators

The refinement order remains superset. The operators previously introduced can be lifted to the traces/successes/failures model by defining their failures, and several more operators are now meaningful. In particular, the first approximation to a recursion is now **div** rather than **stop**.

$\text{failures}(\textbf{stop}) = I^*\,.$

$\text{failures}(P_0 \sqcap P_1) = \text{failures}(P_0) \cup \text{failures}(P_1)\,.$

$$\begin{aligned}&\text{failures}(t \;\rightarrow\; P)\\ &= (I^* \setminus \{u, v, w \;:\; \{u, v\} \subseteq I^* \wedge w \in O^* \wedge vw \ltimes tu \;:\; v\})\\ &\quad \cup \{u, v \;:\; u \in \text{failures}(P) \wedge v \ltimes tu \;:\; v\}\,.\end{aligned}$$

A guarded choice is between those prefixed-processes for which all the inputs required by their guards are available.

$$\begin{aligned}&\text{failures}((|_{0 \le i < n} t_i \;\rightarrow\; P_i)\\ &= (I^* \setminus \{i, u, v, w \;:\; 0 \le i < n \wedge \{u, v\} \subseteq I^* \wedge w \in O^* \wedge vw \ltimes t_i u \;:\; v\}\\ &\quad \cup \{i, u, v \;:\; 0 \le i < n \wedge u \in \text{failures}(P_i) \wedge v \ltimes t_i u \;:\; v\}\,.\end{aligned}$$

$\text{failures}(\textbf{skip}) = \text{failures}(t^{-1}) = \emptyset\,.$

$\text{failures}(t) = I^* \setminus \{u, v, w \;:\; \{u, v\} \subseteq I^* \wedge w \in O^* \wedge vw \ltimes tu \;:\; v\}\,.$

$\text{failures}(P/t) = \{u : tu \in \text{failures}(P) : u\}\,.$

$$\begin{aligned}&\text{failures}(P_0; P_1)\\ &= \text{failures}(P_0)\\ &\quad \cup \{t_0, t_1, t, v : (t_0, v) \in \text{successes}(P_0) \wedge vt_1 \in \text{failures}(P_1) \wedge t \ltimes t_0 t_1 : t\}\,.\end{aligned}$$

$$\begin{aligned}&\text{failures}(P_0 \parallel P_1)\\ &= \{t : t \in (A_0 \cup A_1)^* \wedge (\; (t \upharpoonright A_0 \in \text{dom}(\text{successes}(P_0)) \wedge t \upharpoonright A_1 \in \text{failures}(P_1))\\ &\qquad\qquad \vee\; (t \upharpoonright A_0 \in \text{failures}(P_0)) \wedge t \upharpoonright A_1 \in \text{dom}(\text{successes}(P_1))\\ &\qquad\qquad \vee\; (t \upharpoonright A_0 \in \text{failures}(P_0) \wedge t \upharpoonright A_1 \in \text{failures}(P_1))\;) \;:\; t \setminus C\}\,.\end{aligned}$$

Divergence. The process **div** never becomes stable.

$\text{traces}(\textbf{div}) = I^*$

$\text{successes}(\textbf{div}) = \text{failures}(\textbf{div}) = \emptyset\,.$

Angelic choice. The process $P_0 \Box P_1$ [6] behaves like P_0 or like P_1, except that it can only refuse to produce its *first* output if both P_0 and P_1 can so refuse. (In CSP, a similar operator can only refuse to engage in its first event if both arguments can so refuse.)

$\text{traces}(P_0 \Box P_1) = \text{traces}(P_0) \cup \text{traces}(P_1)$

$\text{successes}(P_0 \Box P_1) = \text{successes}(P_0) \cup \text{successes}(P_1)$

[6] Unfortunately, Broy [5] uses the symbol ∇ for angelic choice and $\Box$ for erratic choice.

$$\mathrm{failures}(P_0 \Box P_1) = ((\mathrm{failures}(P_0) \cup \mathrm{failures}(P_1)) \setminus I^*) \cup (\mathrm{failures}(P_0) \cap \mathrm{failures}(P_1))\,.$$

The following 'step' law [27] relates angelic choice to guarded choice.

$$\boxed{\begin{array}{l}\text{Let } Q = |_{0 \le i < m} t_i \rightarrow P_i \text{ and } R = |_{m \le i < n} t_i \rightarrow P_i. \text{ Then} \\ Q \Box R = |_{0 \le i < n} t_i \rightarrow ((P_i \Box ((R \lhd i < m \rhd Q)/t_i)) \lhd t_i \in I^* \rhd P_i)\,.\end{array}}$$

6 Successes/Failures/Divergences Model

One may object to the above models on the grounds that they imply that divergence is the best thing that can happen. An alternative approach goes to the opposite extreme, as in Brookes and Roscoe's failures/divergences model [4]. Divergence is modelled as chaos, being the worst thing that can happen. The set D of divergences is a subset of A^*.

In the traces/successes/failures model, T had to be given explicitly because after certain traces no stable state might be reachable. Now, however, we shall consider every divergence to be a failure too, enabling us to extract T from F and S, by means of the following definition:

$$T = \{t, u : u \in O^* \wedge tu \in F \cup \mathrm{dom}(S) : t\}\,.$$

All nine conditions of the previous sections must still be satisfied. There is some redundancy, however, since Conditions (4), (5) and (8) follow from the definition of T and Conditions (7) and (9).

6.1 Healthiness conditions

There are five more healthiness conditions:

Every divergence is a failure.

$$D \subseteq F \tag{10}$$

Every divergence paired with any sequence of input events is a success.

$$D \times I^* \subseteq S \tag{11}$$

D is extension-closed.

$$DA^* \subseteq D \tag{12}$$

D is closed under reordering.

$$\{t, u : t \in D \wedge u \ltimes t : u\} \subseteq D \tag{13}$$

Every trace that gives rise to unbounded nondeterminism is a divergence.

$$\begin{aligned}\{t : &|\{u : u \in O^* \wedge tu \in F : u\}| \\ &+ |\{u, v : u \in O^* \wedge (tu, v) \in S : (u, v)\}| = \infty : t\} \subseteq D\end{aligned} \tag{14}$$

Note that Conditions (10), (12) and (14) together imply that $\{t, u : u \in O^* \wedge tu \in D : t\} \subseteq D$. Note also that, if $O \neq \emptyset$, then D can be defined to be

$\{t : |\{u : u \in O^* \wedge tu \in F : u\}| + |\{u, v : u \in O^* \wedge (tu, v) \in S : (u, v)\}| = \infty : t\}$, Condition (13) becoming redundant because it follows from the definition of D and Conditions (7) and (9).

This time the conditions that we impose mean that our space of healthy tuples is not a complete lattice, but only a complete partial order (cpo) under the component-wise superset order.

6.2 Operators

The refinement order remains component-wise superset. Recursion is now the least fixed point with respect to this order.

As T, F and S have to be augmented to reflect the chaos that arises after divergence, we need new semantic functions $\text{traces}_\perp(P)$, $\text{successes}_\perp(P)$, $\text{failures}_\perp(P)$ and $\text{divergences}(P)$.

The semantic clauses for each operator will be omitted here, but it is worth making the following point. In the successes/failures/divergences model, if $t \in O^*$, then $(t \rightarrow \textbf{div}) = \textbf{div}$ and so $\mu X. (t \rightarrow X) = \textbf{div}$. Thus the model is unsuitable for analysis of networks relying upon processes that output forever without waiting for input. The traces/successes/failures model should be used instead.

7 Conclusion

A mathematical framework has been developed for Data-Flow Sequential Processes, a CSP-like language that assumes buffered communication between processes. DFSP operators include **stop**, **skip**, **div**, recursion, reaction, unreading, after, nondeterministic choice, angelic choice, conditional choice, prefixing, guarded choice, sequential composition, iteration and parallel composition. Each term in DFSP denotes a process in an abstract model. The traces/successes/ failures model is an adaptation of the stable-failures model of CSP; the successes/ failures/divergences model is an adaptation of the failures/divergences model of CSP.

Upon termination of a process, the contents of its input streams and output streams remain available to its successor. It suffices to know how that successor behaves for the single case in which all of the streams are empty, in order to determine the effect of this sequential composition! This contrasts with the semantics of occam [26], for example. Stubs allow state information to be passed through a sequence of processes. Alternatively, Broy and Lengauer's approach [6] is adequate for nondeterministic choice, but does not handle angelic choice (as found in 'nonstrict merge', for example) [5]. Failures-based models are more expressive in this respect.

Note that program variables can be accommodated within DFSP: each variable is represented by an input stream on which read and unread actions are performed; their states are passed through stubs.

This paper has highlighted just a few of the algebraic laws of DFSP, focusing instead on its denotational semantics. As future work one might follow the same research agenda for DFSP as for CSP [27], namely, algebraic semantics, operational semantics, relationships between the various semantics, a full-abstraction result for the denotational models, a characterization of deterministic processes, application of the theory and provision of tool support.

Acknowledgment

The author is grateful to the anonymous reviewer for his insightful comments.

References

1. J.D. Brock, W.B. Ackerman. Scenarios: A Model of Non-Determinate Computation. In J. Dia, I. Ramos (editors) *Formalization of Programming Concepts*, Lect. Notes in Comp. Sci. **107**, pp. 252–259, Springer-Verlag, 1981.
2. S.D. Brookes. On the Kahn Principle and Fair Networks. Technical Report CMU-CS-98-156, School of Computer Science, Carnegie Mellon University, Pittsburg, USA, 1998.
3. S.D. Brookes. Traces, Pomsets, Fairness and Full Abstraction for Communicating Processes. In L. Brim, P. Janar, M. Ketinsky, A. Kuera (editors) *CONCUR 2002 — Concurrency Theory*, Lect. Notes in Comp. Sci. **2421**, pp. 466–482, Springer-Verlag, 2002.
4. S.D. Brookes, A.W. Roscoe. An improved failures model for CSP. In S.D. Brookes (editor) *seminar on Semantics of Concurrency*, Lect. Notes in Comp. Sci. **197**, pp. 281–305, Springer-Verlag, 1985.
5. M. Broy. A Theory for Nondeterminism, Parallelism, Communication, and Concurrency. *Theoretical Computer Science* **45**, pp. 1–61, 1986.
6. M. Broy, C. Lengauer. On Denotational versus Predicative Semantics. *Journal of Computer and System Sciences* **42**(1), pp. 1–29, 1991.
7. K.M. Chandy, J. Misra. Reasoning about networks of communicating processes. *Unpublished.* Presented at INRIA Advanced NATO Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, La Colle-sur-Loupe, France, 1984.
8. J.L. Gischer. The Equational Theory of Pomsets. *Theoretical Computer Science* **62**, pp. 299–224, 1988.
9. He Jifeng, M.B. Josephs, C.A.R. Hoare. A Theory of Synchrony and Asynchrony. In M. Broy, C.B. Jones (editors) *Programming Concepts and Methods*, pp. 459–478, Elsevier Science Publishers (North-Holland), 1990.
10. E.C.R. Hehner. Predicative Programming Part II. *Communications of the ACM* **27**(2), pp. 144–151, 1984.
11. C.A.R. Hoare. A model for communicating sequential processes. In R.M. McKeag, A.M. MacNaughten (editors) *On the construction of programs*, pp. 229–254, Cambridge University Press, 1980.
12. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
13. C.A.R. Hoare, He Jifeng. *Unifying Theories of Programming.* Prentice Hall, 1998.
14. B. Jonsson. A model and proof system for asynchronous networks. Proc. 4th Annual ACM Symp. on Principles of Distributed Computing, pp. 49–58, 1985.

15. M.B. Josephs, Receptive process theory. *Acta Informatica* **29**, pp. 17–31, 1992.
16. M.B. Josephs, C.A.R. Hoare, He Jifeng. A theory of asynchronous processes. Technical Report PRG-TR-6-89, Oxford University Computing Laboratory, Oxford, England, 1989.
17. G. Kahn. The semantics of a simple language for parallel programming. In J.L. Rosenfeld (editor) *Information Processing '74*, pp. 471–475, North-Holland, 1974.
18. G. Kahn, D.B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist (editor) *Information Processing '77*, pp. 993–998, North-Holland, 1977.
19. P.G. Lucassen. A Denotational Model and Composition Theorems for a Calculus of Delay-Insensitive Specifications. PhD Thesis, University of Groningen, Groningen, The Netherlands, 1994.
20. W.C. Mallon. Theories and Tools for the Design of Delay-Insensitive Communicating Processes. PhD Thesis, University of Groningen, Groningen, The Netherlands, 2000.
21. A. Mazurkiewicz. Concurrent Program Schemes and their Interpretation. Technical Report DAIMI PB-78, Århus University, Denmark, 1977.
22. E.-R. Olderog, C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informatica* **23**, pp. 9–66, 1986.
23. P. Panangaden, V. Shanbhogue. The Expressive Power of Indeterminate Dataflow Primitives. *Information and Computation* **98**, pp. 99–131, 1992.
24. V.R. Pratt. On the composition of processes. Proc. 9th Annual ACM Symp. on Principles of Programming Languages, pp. 213–223, 1982.
25. V.R. Pratt. Modeling Concurrency with Partial Orders. *International Journal of Parallel Programming* **15**(1), pp. 33–71, 1986.
26. A.W. Roscoe. Denotational semantics for occam. In S.D. Brookes, A.W. Roscoe, G. Winskel (editors) *Seminar on Concurrency, Lecture Notes in Computer Science* **197**, pp. 306–321, Springer-Verlag, 1984.
27. A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.
28. J.T. Udding. Classification and Composition of Delay-Insensitive Circuits. PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1984.