Process Constructors and Interpretations (Invited Paper).

INFORMATION PROCESSING 86, H.-J. Kugler (ed.)
Elsevier Science Publishers B.V. (North-Holland)

PROCESS CONSTRUCTORS AND INTERPRETATIONS

Robin MILNER
Department of Computer Science, University of Edinburgh
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, U.K.

Invited Paper

1. INTRODUCTION

In recent years it has become clear that sequential models of computation, though strikingly successful in explaining the working of a single (sequential) computer program and the way in which such programs may be composed of parts - e.g. procedures - while maintaining a single locus of control, are seriously deficient in explaining how a heterarchic assembly of computing agents behave together. The pioneer in breaking clear from such sequential models was Carl Petri. It is a tribute to the insight shown in his Net Theory that, although many different models of concurrency have emerged in the last ten or fifteen years, they do not supersede his basic ideas - which are more than twenty years old - but only extend them.

However, the aim of this paper is to examine certain basic concepts of communication which are not prominent in Net Theory; this is because they are primarily to do with the way in which we build bigger systems from smaller ones which interact. A large part of computing science is bound up with the study and use of such primitive constructions, both because of their importance and because of the difficulty in choosing them. Their importance probably needs no further comment, but it is worth a moment to ask why they are so difficult to choose. One reason is obvious: it is hard to choose primitive concepts because, by definition, there is nothing from which to build them - there are merely criteria for determining whether their choice is successful. And this leads naturally to a second reason: the criteria for measuring their success are often in conflict. It is illuminating to take primitive-recursive computation as an example. Using the criterion of theoretical understanding, the smallest set of primitive constructions (composition, selection and the primitive-recursion schema) is the best to choose, and has been as successful as one can imagine. But using the criterion of practical utility - and this example is not erudite because most programs do indeed compute primitive-recursive functions over a variety of data types - one would never choose just these constructions as the basis for a programming language. We cannot necessarily expect to find a set of constructions which form a useful practical tool (and we would like to consider calculi and specification languages as tools, as well as programming languages) by simply defining some composite constructions from the "theoretical" primitives. In the case of primitive-recursive computation, a practical tool-kit comprising assignment statements, conditional statements and for statements presents a seriously different model, and the translation to the theoretical model is both non-trivial and illuminating.

The growth of understanding of interacting systems seems to be inseparable from the activity of proposing primitives, assessing them by both theoretical and pragmatic criteria, and attempting to translate between different sets of primitives. When two sets of primitives are incomparable (neither can be translated to the other) then either we must assume that the two models are incompatible - they model essentially different things - or, more positively, we succeed in finding a more elementary (but not larger!) set to which both may be reduced. The following section is an exercise in this type of investigation; we set up principles which guide the choice of primitive constructors, and arrive at two alternative sets which promise to be sufficiently rich in expression. In the third section, we compare and contrast some interpretations of process constructions. The Petri Net interpretation offers a good chance of answering the question "What is a process?", thus clarifying what we should mean above by the phrase "sufficiently rich" (namely: sufficient to express all processes); on the other hand, other interpretations offer at present more tractable mathematical theories.

2. PROCESS CONSTRUCTIONS

In building a model, or in choosing a set of primitive concepts, one must adopt some ideas and principles without prior justification (other than intelligibility), to limit the space of investigation. Just to say that we are trying to model communication is too vague. We shall begin by using the ideas of process and event without definition, and formulate some principles which will guide us to a model at the same time as articulating the meaning of these two terms.

Principle 1 An interaction among processes consists in their participation in a single atomic event.

Though inevitably using the imprecise terms "process" and "event", this principle already excludes some possibilities. For example, the principle denies that a communication via shared memory, as in say Concurrent Pascal, constitutes a single interaction. It also prevents our taking the constructions of Ada as a model, since the Ada rendez-vous primitive involves more than one event in a communication. Further, the principle denies that communication via a buffer or some other medium is an

interaction (again, more than one event is involved). Perhaps these examples also indicate the possibility, which must also be our intention, that many forms of communication will be reducible to the notion of interaction when it is made more precise. The principle does not require all events to be atomic, but for the present paper we shall use "event" to mean "atomic event", where "atomic" means "indivisible in time". Importantly, the principle does not limit the number of processes which may participate in an interaction; thus, although CSP[6] and CCS[9] (as they were originally proposed) represent Principle 1 faithfully, they impose a restriction in allowing only two participants.

Our second principle gives a strong character on our model, since it attributes primary importance to interaction:

Principle 2 Every event is an interaction among processes.

This implies, first, that a computational event such as writing a value into a register must be an interaction, and hence that registers and other media must be treated as processes on a par with the active processes which use them; the effect is to impose a strong homogeneity upon the model. Further, it implies that the observable behaviour of a system (a composite process) consists entirely in its interaction with its environment; the environment may be a human observer who - being also a process - may only inspect or observe the system by interacting with it.

It is now convenient to introduce a set Act = {a,b,...} of actions; an action identifies the external aspect of a process part in an event, while the internal aspect is a state-transition by the process. Further, we shall use P,Q,... to stand for processes, where we use the term "process" as equivalent to "process-in-a-state". Then we may write

$$P \xrightarrow{a} P'$$

to mean that the process (in the state) P may perform the action "a", and simultaneously become the process (in the state) P'. Thus, in view of Principle 2, once the transition relations $\xrightarrow{a}$ are known over processes, we know all about the behaviour of every process.

But our theory is, up to this point, embarrassingly poor! The reason is that we have given no structure either to the actions or to the processes, and (since structure is essential for understanding systems) we are far from having a useful model. Structure will follow shortly, but an important point must be made - and stands out all the more clearly before we become preoccupied with structure. If a process P has the following transitions

$$P \xrightarrow{a} P_1 \xrightarrow{b} P_2$$

$$P \xrightarrow{b} P_3 \xrightarrow{a} P_4$$

then - without knowing the structure of P - can we determine whether the actions "a" and "b" are causally independent (= "concurrent" in Net Theory) or not, since they can occur in either order? If we cannot, then causal independence among actions is not an attribute of P's behaviour, i.e. it is intensional, not extensional. This will be so even if the structure of Act admits a composite action "(ab)", meaning "a" and "b" simultaneously; the presence of another transition

$$P \xrightarrow{(ab)} P_5$$

might simply represent a third alternative for P, and these three alternatives taken together do not necessarily add up to the causal independence of "a" and "b". Though this argument is not conclusive, the view that observation is interaction appears to commit us also to the view that causality is not an attribute of behaviour (not observable), but - however useful in analysis - is rather a property of the structure of processes. We may express this another way: the behaviour of a process does not reveal its composition, and thus a simple sequential process may exhibit the same behaviour as a complex process with many independent parts. In a recent paper [12] Wolfgang Reisig has examined the relationship between transition systems and concurrency.

Now that we have begun to consider structure, we look for constructors over Proc which yield complex processes from simple ones. For the remainder of this paper we shall use the term "constructor" to mean a basic operation for building processes, and the term "construction" to mean a possibly complex operation which is a composition of constructors.

Principle 3 Every n-ary Process constructor f must be such that the behaviour of $f(P_1,\ldots,P_n)$ depends only upon the behaviours of $P_1,\ldots,P_n$.

We must admit that "behaviour" is still an imprecise notion, except that we have taken it (following Principle 2) to depend only upon actions, the external aspect of transitions. Thus it is natural to tighten Principle 3 to require that the actions of $f(P_1,\ldots,P_n)$ depend only upon those of $P_1,\ldots,P_n$. Perhaps the most natural constructor, which may be called conjunction (&) may be described as follows: P & Q has exactly the transitions $P \& Q \xrightarrow{a} P' \& Q'$ for which both $P \xrightarrow{a} P'$ and $Q \xrightarrow{a} Q'$. We shall write this condition as a rule:

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \,\&\, Q \xrightarrow{a} P' \,\&\, Q'}$$

(For each constructor we shall provide one or more such rules, implying that its transitions are exactly those which follow from the rules). The conjunction of two processes exhibits exactly that behaviour which is common to them both; they are required to synchronise upon every single action. Moreover, iterated conjunction $P_1 \& P_2 \& \ldots \& P_n$ causes n processes to synchronise in this way.

This constructor allows the component processes no independence of action, and is too crude to model interesting interactive behaviour. We need to combine processes in such a way that they sometimes interact and sometimes act independently.

Principle 4 (Conjunction) A construction is needed which will force each of n processes to synchronise with all the others upon any of a designated set of actions, but to perform all other actions freely.

This construction can be achieved by iterating a binary constructor $\&_A$, where A is the designated set of actions. The rules which define $\&_A$ are as follows:

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \,\&_A\, Q \xrightarrow{a} P' \,\&_A\, Q'} \; (a \in A)$$

$$\frac{P \xrightarrow{b} P'}{P \,\&_A\, Q \xrightarrow{b} P' \,\&_A\, Q} \; (b \notin A) \qquad \frac{Q \xrightarrow{b} Q'}{P \,\&_A\, Q \xrightarrow{b} P \,\&_A\, Q'} \; (b \notin A)$$

Note that Principle 4 implies asynchrony; one member of a collection of processes may perform an action (not in the designated set) while others remain dormant. This possibility is achieved by the last two rules for $\&_A$, while synchronisation is achieved by the first rule. Constructors of this kind, which depend upon explicit designation of the synchronising actions, have been adopted both by Hoare [7] and Milne [8]; we discuss later the precise form which they adopt.

The &-constructors still lack something. Since each action "a" of a component is also an action of the construction, each interaction among the components is also observable from outside.

Principle 5 (Encapsulation) A construction is needed which renders a designated set of actions unobservable.

The simplest way to meet this need is by the unary hiding constructor -/A, where $A \subseteq Act$ is the designated set. It depends upon a distinguished member e of Act; e may be thought of as the perfect event, needing no further participation (even from an external observer) for its occurrence.

$$\frac{P \xrightarrow{a} P'}{P/A \xrightarrow{e} P'/A} \; (a \in A) \qquad \frac{P \xrightarrow{b} P'}{P/A \xrightarrow{b} P'/A} \; (b \notin A)$$

Let us henceforward write $P \,\&_A\, Q$ for $P \,\&_{\{a\}}\, Q$, and P/a for $P/\{a\}$. As an example, the construction $(P \,\&_a\, Q)/a \;\&_a\; (R \,\&_a\, S)/a$ will enforce (and encapsulate) interaction between P and Q on "a", and between R and S, but not between the two pairs.

Note that the distinctions of e, the perfect event, is our first commitment to some structure over Act, the action set; we find a use for further structure later.

Are these two families of constructors, $\&_A$ and -/A enough? They have considerable power, if we use different designated sets A. Take, for example, the construction $P \,\&_a\, (Q \,\&_\phi\, R)$; it contains an element of disjunction, as well as conjunction, for it ensures that P interacts either with Q or with R upon "a", but not with both simultaneously. But there are richer possibilities which cannot be realised. Consider a trio (P,Q,R) of processes, where we require each member to interact with either, but not both, of the others upon the action "a". With a little more formality, it is fairly easy to prove that no construction C(P,Q,R) which can be built using conjunction and hiding alone can meet this requirement.

It is quite hard to formulate a principle of disjunction which is properly general and subsumes the last example as a particular case. For the present paper we will be content to put forward the following.

Principle 6 (Disjunction) A construction is needed which will force each of n processes to synchronize with any one of the others upon any of a designated set of actions.

In view of the above example, to satisfy this principle we must either add a constructor or replace the existing ones. We consider the former alternative first, since a further principle can be satisfied by a constructor which also permits the construction demanded by Principle 6. The point of this further principle is that the names of actions should not be immutable.

Principle 7 (Renaming) A construction is needed which changes the action-names of a process.

This can be met by introducing the constructor $-[\Phi]$ for any binary relation Φ over Act. Its rule is simple:

$$\frac{P \xrightarrow{a} P'}{P[\Phi] \xrightarrow{b} P'[\Phi]} \; (a \,\Phi\, b)$$

(Note: in a precise treatment one should stipulate that the perfect event e is never synchronised, hidden or renamed!). We leave it as an exercise for the reader to build the construction C(P,Q,R) above, using conjunction and (many-valued) renaming. As another interesting exercise, he or she may like to show that the hiding construction P/A is exactly the same as the renaming construction P[e/A], where "e/A" replaces each member of A by e, leaving all other actions unchanged.

We conclude that, together, conjunction $\&_A$ and renaming $[\Phi]$ provide sufficient power of expression to satisfy Principles 4-7, and are consistent also with Principles 1-3. Before looking at alternatives, we state another principle which ensures that we may consider our constructors as a language - whether for programming, or describing, or specifying distributed processes.

Principle 8 Constructions are needed for sequential control, rich enough to express a wide range of distributed processes.

This principle compensates for the emphasis placed upon parallel composition by the preceding principles. It turns out that rather simple constructors for sequential composition, alternative action and repetition are sufficient. We may take for example a.P to mean the prefixing of a single action "a" to P, P + Q to mean the process which may behave like P or Q (the choice being determined by the first act-

ion) and recX.-X- (where X is a process variable) to achieve repetition by recursion. These constructors have simple rules of action; see [9],[10],[5].

The reader may wonder why Principle 8 contains the vague expression "a wide range of distributed processes". The reason is uncomfortable; we have yet no firm idea of what a distributed process *is*, in the same way as we know what a (computable) function is, so we cannot yet know what it requires to express them all! We shall return to this point in the next section.

Now let us return to consider alternatives for conjunction and renaming. These constructors stand up fairly well to the pragmatic criterion that it should be convenient to write process descriptions or specifications. They stand up less well, perhaps, to the criterion of theoretical analysis. One essential property of an appropriate theory is a simple but powerful repertoire of equational laws (valid for whatever notion of equality is adopted for process expressions), to allow transformations which preserve meaning. Some such laws, for example $P \&_A (Q \&_A R) = (P \&_A Q) \&_A R$ (associativity of conjunction) are readily obtained. But the algebraic theory is complicated by the presence of an indexed family of binary constuctors, $\&_A$. For example, what equivalent forms has $P \&_A (Q \&_B R)$ for arbitrary $A,B \subseteq Act$? An infinity of binary operators in an algebraic theory is likely to cause serious complication, and this case is no exception.

Hoare [7] and Milne [8] have found one solution to this problem. They first demand that to each process P is associated an alphabet $\alpha(P) \subseteq Act$; then they adopt a single parallel constructor ("||" for Hoare, "·" for Milne) which requires its two argument processes to synchronise on any action common to their alphabets. Its rules of action, in our current framework, are as follows:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P \| Q \xrightarrow{a} P' \| Q'} (a \in \alpha(P) \cap \alpha(Q))$$

$$\frac{P \xrightarrow{b} P'}{P \| Q \xrightarrow{b} P' \| Q} (b \notin \alpha(Q)) \qquad \frac{Q \xrightarrow{b} Q'}{P \| Q \xrightarrow{b} P \| Q'} (b \notin \alpha(P))$$

(Milne's constructor also allows actions which consist of several simultaneous "particulate" actions, similar to an idea which we discuss a little later.)

This constructor has nice algebraic properties such as associativity, as Hoare has shown, and avoids the ocmplications of an infinite family of constructors. There is one difficulty; the alphabet of a process P cannot be deduced only from the expression of P by constructors, but must be specified independently. For example, the algebraic law L4A on p67 of [7]

$$a.P \| a.Q = a.(P \| Q)$$

(which appears obvious, perhaps) only holds when the same alphabet is assigned to a.P and P, and likewise to a.Q and Q.

One of the aims in choosing constructors for CCS [9] was also to achieve a single parallel constructor, without dependence upon the alphabet assigned to its arguments. (Some of the equational laws of CCS do indeed depend on the *sort* of a process, but the sort is an alphabet which is derivable from the process expression). However, a different price was paid - as we shall see below.

To define this constructor, we assume that $e \in Act$ and that there is a bijection $\bar{}$ over Act-{e} such that $\bar{\bar{a}} = a$; the bijection is called *complementation*.

Now, the single binary constructor $|$, process *composition*, is defined by the following rules:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{e} P'|Q'} (a \neq e)$$

$$\frac{P \xrightarrow{a} P'}{P|Q \xrightarrow{a} P'|Q} \qquad \frac{Q \xrightarrow{a} Q'}{P|Q \xrightarrow{a} P|Q'}$$

It can be shown that Principle 6 (Disjunction) is satisfied by $|$ together with other (unary) constructors; this is because the rules ensure that any action "a" ($\neq e$) synchronises with its complement but not with itself.

Further, because $|$ imposes no constraint upon the independent action of component processes (since there is no side-condition in its second and third rules), it needs to be accompanied not by the *hiding* constructor -/A (which perfects an action by releasing it from the need for further participation) but by the *restriction* constructor -\A (which prevents imperfect occurrence of any member of A):

$$\frac{P \xrightarrow{b} P'}{P\backslash A \xrightarrow{b} P'\backslash A} (b \notin A)$$

In fact hiding is expressible in terms of composition and restriction; for we may assume by Principle 8 that the process a^ω, whose only action is $a^\omega \xrightarrow{a} a^\omega$, is expressible, and we can easily see that

$$P/a = (P|a^\omega)\backslash a$$

A similar expression exists for the general form P/A.

However, the constructor $|$, together with other (unary) constructors, does not satisfy Principle 4 (Conjunction). The reason is that, by the first rule of composition, only two processes may be synchronised; the result of this synchronisation is the perfect event e, which is not susceptible to synchronisation with further processes.

A way out of this difficulty is possible, which at the same time will satisfy a further principle which has wider claim to adoption:

Principle 9 (Simultaneity) The simultaneous occurrence of two actions is also an action.

This may be supported as follows. At a keyboard, I may be able to interact with a process P by depressing a key labelled "a"; consider this as my observation of "a", or as my contrib-

uting "$\bar{a}$" to an event. I may also be able to interact with Q by depressing "b" (contributing "$\bar{b}$"). By linking the two keys, I can be sure to contribute "$\bar{a}$" and "$\bar{b}$" simultaneously (or to observe "a" and "b" simultaneously, or to interact with P and Q simultaneously). Thus, my contribution of "$(\bar{a}\bar{b})$" will be to a perfect event which is a three-way synchronisation of P,Q and myself. We can model all this by introducing a binary operation × (product, or synchronisation) over Act, and by stipulating that $(Act,\times,^{-},e)$ is a commutative group. This step was taken in [2] to yield a synchronous version of CCS; but in that paper it was not made sufficiently clear that it provides an asynchronous process model which stands in its own right, and which allows all the principles which we have stated to be satisfied by just a single binary constructor (an extension of $|$) and the indexed family $-\backslash A$ of unary constructors (plus constructors for sequential control).

In fact, the rules for composition reflect a partial product of actions, defined only between complementary actions: $(a\bar{a}) = e$. (We shall generally write (ab) in place of $a \times b$). The extension to a total product is therefore naturally matched by an extension to the first rule for $|$; the rules for composition now become

$$\frac{P \xrightarrow{a} P \qquad Q \xrightarrow{b} Q'}{P|Q \xrightarrow{(ab)} P'|Q'}$$

$$\frac{P \xrightarrow{a} P'}{P|Q \xrightarrow{a} P'|Q} \qquad \frac{Q \xrightarrow{a} Q'}{P|Q \xrightarrow{a} P|Q'}$$

To justify our claims, it is convenient (perhaps necessary) to assume that Act is the free commutative group generated by a set Part = {p,q,...} of particles, or particulate actions (see [10]). Thus every action "a" is a unique product of non-zero powers of distinct particles, $a = p_1^{z_1} \ldots p_k^{z_k}$. (If $k = 0$ then $a = e$, and for negative powers we take p^{-n} to mean $\bar{p}^{n}$). If "a" and "b" do not contain the same particle, then we write a#b, and if a#b for all $b \in A$ we write a#A. Then we consistently redefine the rule for $-\backslash A$ as follows:

$$\frac{P \xrightarrow{a} P'}{P\backslash A \xrightarrow{a} P'\backslash A} \ (a\#A)$$

(Note that e#A always holds, so restriction cannot prevent the perfect event from occurring!). We may now express conjunction $\&_A$ in terms of composition, restriction and renaming as follows (we consider only the case that A contains a single particle p):

$$P\&_p Q = (P[p_1/p] \,|\, Q[p_2/p] \,|\, (\bar{p}_1\bar{p}_2 p)^{\omega})\backslash\{p_1,p_2\}$$

where p_1 and p_2 are chosen not to occur in P or Q. To see this note that the triple product $\bar{p}_1\bar{p}_2 p$, together with the encapsulation of p_1 and $\bar{p}_2$, ensures that the composite process performs the action $p_1p_2\bar{p}_1\bar{p}_2p = p$ when, and only when, both P and Q perform p simultaneously.

Finally, the renaming of one particle by another can be defined using only composition and restriction:

$$P[q/p] = (P \,|\, (\bar{p}q)^{\omega})\backslash p$$

and more general (even multivalued) renamings are readily definable.

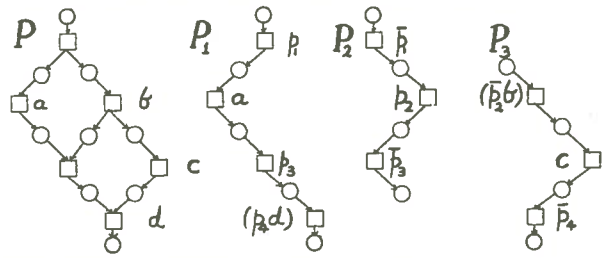
The relative merits of the various parallel constructors can only be assessed by working with them both theoretically and practically, and we cannot perform this task here. Hoare points out, in particular, that $\&_A$ and $\|$ have the advantage that they preserve the property of determinacy of a process, which $|$ does not. This is certainly a point in their favour. But most realistic process constructions will also use some form of encapsulation - either hiding or restriction - and it can be easily seen that the hiding constructor $-/A$ does not preserve determinacy in general. Thus, because hiding is the form of encapsulation needed to accompany $\&_A$ or $\|$, the weight of the advantage is not too clear.

To summarise this section: we have described two alternative sets of constructors which - to the extent that our principles carry weight - promise to express much of what we intend by the indistinct notion of "process". By examining these constructors in greater depth, and in particular by asking when two compound constructions are plausibly equivalent, we may hope to make the notion more distinct.

Process Interpretations

We now wish to look at different ways of interpreting process constructions. We do not hope to arrive at a best interpretation. Rather, we point out some striking conceptual differences among the alternatives.

The interpretation which is probably the most refined, in that it makes the fullest distinction which we would like to admit among our constructions, is in terms of Petri Nets, where we shall label the event nodes by actions. We omit the label e, the perfect event; other action labels indicate potential events, or - in our terminology - actions which are imperfect but may be perfected by interaction. Consider the following condition/event net, shown with a natural decomposition into linear nets:



Now, using constant 0 for the process which is incapable of action, we can naturally express P as follows:

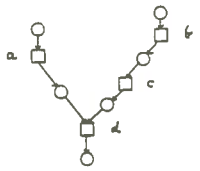
$P=(P_1|P_2|P_3)\backslash\{p_1,p_2,p_3,p_4\}$ where $P_1=p_1.a.p_3.(p_4d).0$,

$P_2=\bar{p}_1.p_2.\bar{p}_3.0$, $\quad P_3=(\bar{p}_2 b).c.\bar{p}_4.0$

Of course there are many such decompositions of P, each expressed by a different construction. It is therefore a challenge - which I believe has not been met - to find an equational theory

of processes in which two constructions may be proved equal exactly when they denote the same net; this seems difficult even for finite processes!

The problem is even harder if we wish to equate nets which only differ in (certain) occurrences of the perfect event. For example, we may wish to equate P with the following net P', gained by omitting two occurrences of the perfect event:



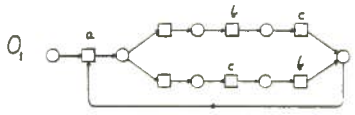
Some steps in this direction are surveyed in [11]; however, even the most refined of these equivalences (close to observational equivalence discussed below) do not respect causal independence among events.

Indeed, the outstanding advantage of the net interpretation is that concurrency, or causal independence, among actions is very clearly represented. If the concept of concurrency is taken to be intrinsic in the notion of process, then Net Theory perhaps offers us the way to answer the question which was raised in the previous section: what is a computable process? A possible answer is that computable processes are exactly those recursively enumerable acyclic nets which obey certain natural conditions (for example, that the in-degree and out-degree of nodes should be finite). This answer would be strongly reinforced by a theorem which states that this class is exactly expressible by one or another set of constructors as proposed in the previous section; the choice of these constructors would in turn receive justification.

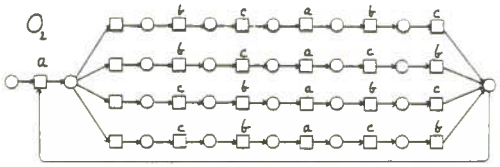
The above examples of nets are purely causal; they contain no non-determinism, since in Net Theory the arbitrary order of occurrence of causally independent events does not count as non-determinism. By contrast with Net Theory, several equivalences have been proposed for our process constructions which do not preserve causal independence; rather, they equate causal independence with arbitrary occurrence order, on the grounds that an observer or experimenter cannot detect causality. Such equivalences are observational equivalence [9]. refusals equivalence [2] and testing equivalence [4],[3]. These equivalences differ, but have one thing in common; the observer is understood to be capable of observing only a single action at a time - even if this action is the simultaneous occurrence of two or more particulate actions. It can be persuasively argued that the notion of causality cannot be based upon such an observer's experience, and that an observer-based notion of process is therefore defective. The behaviourist may reply, also persuasively, that only what is observable (or extensional) is real, and that the notion of process should therefore not represent causality; rather, causality - though a convenient analytical tool - is best regarded as an intensional property of a presentation (or expression) of a process. Be that as it may, the observer-based equivalences do have a strong advantage; as their proponents have shown, they all enjoy pleasant equational theories which are complete at least for finite processes, and even for larger classes of process. Thus we see a real trade-off between refinement of interpretation on the one hand, and power of reasoning on the other hand.

We now turn to an example which will serve to distinguish between some of the equivalences based upon observation; to be precise, it distinguishes observational equivalence on the one hand from refusals or testing equivalence on the other. The point is not to claim that one equivalence is correct, but rather to show that each has a strong intuitive character, and that the choice of equivalence is therefore a matter to be determined by what is pragmatically acceptable (would we be happy to buy a system Q if it is cheaper than P, and distinguishable from P by one equivalence but not the other?), and by the proof techniques available for each equivalence.

Consider a simple operating system O_1 which can be switched on each morning (action "a") and which will then do your job and my job (actions "b" and "c") in either order, the order being determined by a perfect internal event:



We have presented O_1 as a net, to show that no concurrency exists. Now compare O_1 with a second system O_2 which may still do your job and my job in either order each day, the order being determined by a perfect internal event which occurs only on alternate days:



The expressions for the two systems in CCS are as follows, using recursion:

$$O_1 = a.(e.b.c.O_1 + e.c.b.O_1)$$

$$O_2 = a.(e.b.c.a.b.c.O_2 + \ldots + e.c.b.a.c.b.O_2)$$

Now O_1 and O_2 are not observationally equivalent, since this equivalence demands at least the following for two processes: if one of them can execute a sequence s of actions (ignoring e) in such a way that its set of possible next actions is A, then the other must also be able to execute s so that its set of possible next actions is A. But O_1 may execute s = abca so that its set of possible next actions is A = {b,c}, while O_2 may only execute s so that its set of possible next actions is either A = {b} or A = {c}.

The first interesting equivalence to be proposed which was significantly weaker than observational equivalence was the refusals equivalence of Hoare et al [2]. If R is a set of

actions ($e \notin R$), then R is said to be a refusal of the process P after the action sequence s if P may, after executing s, be in a state where no member of R is a possible action. Now it is easily seen that R is a refusal of 0_1 (resp. 0_2) after s = abca iff $\{b,c\} \not\subseteq R$; once we see this, we can easily prove that 0_1 and 0_2 are refusals-equivalent. A considerable merit of this equivalence relation is that it appears to be the weakest which only equates processes with the same possibility of becoming deadlocked.

We can see the character of refusals equivalence much more dramatically if we observe that both 0_1 and 0_2 are equivalent to a third operating system 0_3 which, at its very outset, makes a sweeping choice (without participation from its user or environment) which determines for every day in the future whether it will perform your job or my job first on that day. In other words, we are led to think of a non-deterministic process as just as a set of deterministic processes. This is Hoare's intention, as he explains in his recent book [7]; we are to regard a non-deterministic process P as a specification, which may be realised by any deterministic process which is a member of P, or perhaps by a less non-deterministic process which is a subset of P. In fact the more ardent adherents of this model are inclined to regard the model of observational equivalence (in which each process is interpreted as its equivalence class under the relation) as presenting too much of a description of each process - revealing its intension, or inner behavior; they argue that it makes distinctions which could not be detected by a natural experimental procedure. One can reply, just as a supporter the Petri Net model might reply, that working with intensional descriptions is valuable in analysis. However, what is intensional and what is extensional is very dependent on what is regarded as an experimental procedure; in a recent paper, Abramsky [1] has shown convincingly that there is a natural hierarchy among experimental procedures, and has even given strong evidence that observational equivalence corresponds exactly to the strongest of all experimental procedures.

The third most notable equivalence of processes is the testing equivalence of De Nicola and Hennessy [4]. This corresponds to a natural experimental procedure; the experimenter behaves as a testing process T, and in each application of test T to an object process P, P may either pass or fail. Then P and Q are said to be testing equivalent exactly when

(1) P sometimes passes T iff Q sometimes passes T.
(2) P always passes T iff Q always passes T.

As an example, to compare the operating systems 0_1 and 0_2 we may apply the linear test $T = \bar{a}.\bar{b}.\bar{c}.\bar{a}.\bar{b}.0$; then (assuming that to pass T is just to allow T to be fully executed) we find that both 0_1 and 0_2 sometimes pass T but do not always pass T; further analysis shows that 0_1 and 0_2 are indeed testing equivalent. De Nicola [3] shows that not all processes need be used as tests; a class of simple finite tests achieves full power of testing.

The difference between refusals and testing equivalence is rather subtle, and in fact not essential; with only minor adjustment (see [3]) they become identical, and this coincidence suggests that a robust notion has been discovered. In fact, though we have no space to discuss it, observational equivalence has also a satisfying alternative characterisation in terms of modal logic [5], besides its more recent experimental characterisation by Abramsky [1].

This discussion of interpretations has been incomplete, and it is certain that other interpretations which we have not mentioned will be useful. But by concentrating upon a few, we have been able to focus attention on some difficult and important issues. One issue, that of complete expressive power, deserves a little more comment. We suggested earlier that an important theorem would state that all computable processes are expressible by one or another constructor set. This is not the only kind of complete expression to look for. Recently Simone [13] has formulated another kind, and proved that the MEIJE calculus (and also a form of CCS) possesses it. More precisely, he shows that any constructor which can be defined by the form of operational rules adopted in the first part of this paper can be expressed - up to a very strong equivalence relation - by the four basic constructors of the calculus. This kind of theorem, asserting complete expressiveness for process constructions rather than processes, should be sought for every calculus.

4. CONCLUSION

We have examined various possible constructions by which complex processes may be built from simpler ones, and by setting up some principles for what should be expressed by these constructions we hope to have given evidence that certain elementary constructors are sufficient to serve as a basis for a theory. We then discussed the interpretation of these constructions, and found that several interpretations have strong claims to consideration. The discussion has certainly omitted some troublesome questions - for example, how divergence (infinite sequences of perfect events) should be treated - and we must admit that many of these questions remain to be properly resolved. Nevertheless, we hope to have shown that a well-knit theory of distributed processes is evolving, in which the different equivalence relations find their place in a hierarchy.

ACKNOWLEDGEMENT

I am most grateful to Tony Hoare for his careful criticism of a draft of this paper. This has improved the presentation and corrected some inaccuracies. But since the views I have expressed are debatable, the reader is urged to consider them in the light of the careful discussions given by Hoare in his book

REFERENCES

1. S. Abramsky, "Observational Equivalence as a Testing Equivalence", Draft paper, Imperial College, 1985.

2. S.D. Brookes, C.A.R. Hoare and A.W. Roscoe,

"A Theory of Communicating Sequential Processes", J.ACM 31 (7) 560-599, 1984.

3. R. de Nicola, Testing Equivalences and Fully Abstract Models for Communicating Processes, Ph.D. Thesis, CST-36-85, Computer Science Dept, University of Edinburgh, 1985.

4. R. de Nicola and M. Hennessy, "Testing Equivalences for Processes", J. Theor. Comp. Sci. 34 (1) 83-135, 1984

5. M. Hennessy and R. Milner, "Algebraic Laws for Nondeterminism and Concurrency", J.ACM 332 (1) 137-162, 1985.

6. C.A.R. Hoare, "Communicating Sequential Processes", Comm. ACM 21 (8), 1978.

7. C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall International, 1985.

8. G.J. Milne, "CIRCAL and the Representation of Communication, Concurrency and Time", ACM Trans. on Prog. Lang. and Sys, 7 (2) 270-298, 1985.

9. R. Milner, A Calculus of Communicating Systems, Vol. 92, Lecture Notes in Computer Science, Springer Verlag, 1980.

10. R. Milner, "Calculi for Synchrony and Asynchrony", J. Theor. Comp. Sci. 25, 267-310, 1983.

11. L. Pomello, "Some Equivalence Notions for Concurrent Systems, an Overview", Proc. 6th European Workshop on Applications and Theory of Petri Nets, Espoo, Finland, 1985.

12. W. Reisig, "What Operational Semantics is Adequate for Nonsequential Systems?", Internal Report, Series B No.30, Helsinki University of Technology, 1984.

13. R. de Simone,"Higher Level Synchronising Devices in MEIJE-SCCS", J. Theor. Comp. Sci. 37 (3) 245-268, 1985.