Theoretical Computer Science 25 (1983) 267–310
North-Holland

CALCULI FOR SYNCHRONY AND ASYNCHRONY*

Robin MILNER

Department of Computer Science, Edinburgh University, Edinburgh EH9 3JZ, United Kingdom

Communicated by M. Nivat
Received February 1982
Revised August 1982

Abstract. A calculus for distributed computation is studied, based upon four combinators. A central idea is an Abelian group of actions which models the interfaces between components of a distributed computing agent. Using a notion of bisimulation, congruence relations are defined over computing agents, and thence an algebraic theory is derived. The calculus models both synchronous and asynchronous computation. In particular, it is shown that the author's Calculus of Communicating Systems (1980), which is an asynchronous model, is derivable from the calculus presented here.

Contents

PART I. SYNCHRONY

1. Introduction

The purpose of the present paper is to bring together, in a single mathematical framework, the modelling of systems which interact both synchronously (i.e. in a time-dependent fashion) and asynchronously. The approach here is entirely

* This research has been supported by a Grant from the Venture Research Unit of the British Petroleum Company.

consistent with CCS, a Calculus of Communicating Systems [8], where only asynchronous interaction was considered, since the problems seemed at first to be simpler in that case. That model is quite appropriate for distributed programs, excepting real-time programs, and to some extent for hardware systems. Time-dependency is a less prominent feature of programs than of hardware, simply because the essential purpose of a programming language is to insulate the programmer from properties of real computers which may clutter his thinking.

It now appears that CCS has a natural generalisation to embrace syncrhonous systems. One outcome of the present work is that, by thinking first of synchronous systems and constructing a calculus for them, we can characterize asynchronous systems as a subclass, and derive an asynchronous calculus. This derivation leads to a rather richer calculus than CCS; however, as we shall show, CCS itself can be derived in a precise sense.

Let us suppose that we have a class $\mathcal{P}$ of *agents*, for which the notion of *atomic action* is understood as follows. There is a set Act of atomic actions, and for each $a \in \text{Act}$ a binary relation $\xrightarrow{a}$ over $\mathcal{P}$; an instance

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

of this relation is to be interpreted "P may perform a, and become P' in doing so". Thus P and P' may be thought of as two states of a system. We call P' a *successor* of P; the derivatives of P are itself and its *proper derivatives*, i.e. the derivatives of its successors.

Let us now assume that time is discrete, and that P existing at time t may, under the above relation instance, become P' at time $t+1$. This is the sense in which actions are atomic; they are indivisible in time. But they may not be indivisible in every sense. We wish to consider a system consisting of two agents P and Q, for which $P \xrightarrow{a} P'$ and $Q \xrightarrow{b} Q'$, to be capable of performing the *product* of the actions a and b instantaneously. Further if $R \xrightarrow{c} R'$, then the system consisting of P, Q and R may perform the product of a, b and c. Now it is natural to assume that product over actions which we shall write '$\times$', is both commutative and associative, since it represents simultaneous occurrence. We therefore postulate

(Act, $\times$) is an Abelian semigroup .

We also wish to consider a composite system of agents to be a single agent. This leads us to require $\mathcal{P}$ to be closed under a binary product operation, also written '$\times$'; from $P \xrightarrow{a} P'$ and $Q \xrightarrow{b} Q'$ we shall infer $P \times Q \xrightarrow{ab} P' \times Q'$. (Product in Act, but not in $\mathcal{P}$, will often be represented by juxtaposition.) Further, we shall require that *every* action of $P \times Q$ can be inferred from actions of the components P and Q; then we shall also deduce that $\mathcal{P}$ itself is an Abelian semigroup.

The latter requirement amounts to assuming that the components of a system proceed synchronously; it is not possible for one component to remain idle for an instant, unless indeed there is an atomic action which corresponds to idling. But it is very natural to admit such an action; it takes its place as a unit action $1 \in \mathrm{Act}$, so we strengthen our postulate to

(Act, ×, 1) is an Abelian monoid

Thus, from $P \xrightarrow{1} P'$ (an idle action) and $Q \xrightarrow{b} Q'$ we can infer $P \times Q \xrightarrow{b} P' \times Q'$. Notice that although the composite action is just that of Q, P has changed to P', and at the next instant P' may not be able to idle An agent P may idle indefinitely long if it possesses the action $P \xrightarrow{1} P$; it may possess other possible actions too, and may perform one of them at any instant. We may think of such agents—and products of them—as *asynchronous*; later we shall make this a precise notion.

Another consequence of our assumptions is that an agent P which possesses no action (not even an idle one) causes 'disaster' in any product, since then no action is possible for $P \times Q$, whatever Q. Such a P constitutes a (unique) zero in our eventual algebra of agents. Certainly it has no direct realisation, but this is no reason to exclude it from our calculus; such unreal agents can play a part in calculating the behaviour of real agents.

In a definitive calculus there should be as few operators or combinators as possible, each of which embodies some distinct and intuitive idea, and which together give completely general expressive power. Though we cannot yet assess any calculus precisely on the second criterion, the present calculus is a distinct advance over [8] towards definitiveness. If we disregard the recursion construction (some such construction is essential for defining infinite behaviour) we have reduced our combinators to the following four, with manifestly distinct rôles:

(i) *Product*, for combining agents concurrently (discussed above);

(ii) *Action*, a combinator representing the occurrence of a single indivisible event;

(iii) *Summation*, the disjunctive combination of agents, allowing alternate courses of action;

(iv) *Restriction*, for delimiting the interface through which an agent interacts with others.

These four operators obey (as we show) several algebraic identities. It is not too much to hope that a class of these identities may be isolated as axioms of an algebraic 'concurrency' theory, analogous (say) to rings or vector spaces. For the present, however, we concentrate on an interpretation of the calculus derived from an operational or dynamic understanding of each operator, whereupon the algebraic identities arise as theorems.

It is certainly tempting to seek models of concurrent computation based upon some mathematical concept of 'process', after the considerable success of Scott and his followers with models of sequential computation based upon the concept of

(continuous) function. The author, among others, has attempted this [7, 13]. Two questions arise for any such construction. First, is it in complete agreement with operational intuition? This criterion, called *full abstraction*, was proposed by the author in [7]; since then, he has come to an even firmer belief that operational semantics, since it can be set up with so few preconceptions, must be the touchstone for assessing mathematical models rather than the reverse. Second, is there a *unique* mathematical concept of process? Each model will capture some aspects of operational behaviour (this will be expressed by its agreement with operational semantics); the problem with nondeterminate concurrent agents—as distinct from determinate sequential ones—is that there are many such aspects. The model of [HBR] for example, while definitely a step forward from [13], equally definitely ignores some aspects, and may be justified in doing so by pragmatic considerations. By contrast, the present approach based purely upon operational definition makes no prior decision on what aspects to ignore, and can admit different mathematical models by employing different operationally-defined congruence relations over agents. The operational approach used is presented in general terms by Plotkin [16], and the author owes much to his development of it over many years.

The present paper is organised in two parts. In the remainder of Part I we first introduce in Section 2 the operational meaning of SCCS, the synchronous calculus. Section 3 presents two simple examples, with proofs which anticipate the use of algebraic laws. In Section 4 we give the interpretation of SCCS as a quotient, by a congruence relation which depends upon the central idea of bisimulation between agents, and establishes a unique fixed point property. The notion of bisimulation is due to David Park [15]. Section 5 contains algebraic laws, and introduces the classification of agents by their sorts, where a sort is a description of the interface presented by an agent to its environment. In Section 6 the expressive power of the calculus is illustrated by some derived constructions, and its essential features are discussed.

Part II begins in Section 7 with a definition of asynchrony, and presents a subcalculus ASCCS in which every agent is asynchronous. Section 8 is concerned with the properties of a weak congruence in ASCCS, based on a weak bisimulation in which agents may be similar even if their time-behaviour differs. Finally, in Section 9 it is shown that CCS, almost exactly as presented in [8], is isomorphic to a subcalculus of ASCCS.

This paper extends and subsumes the author's "On relating Synchrony and Asynchrony" [9]. An outline of the present work appears in [11].

2. A calculus of agents

In our introduction we gave a superficial description of the four combinators of SCCS, which—together with recursion—give us a set of constructions which is small enough to handle theoretically, yet large enough to express real and interesting

composite agents. We now present them formally. For each construction, we follow the principle that its actions should be determined by those of its components.

This procedure amounts to defining a *language*, since we must express our constructions symbolically, and to presenting its operational *meaning* in the form of derivation rules. In fact, our agents are just certain expressions in the language, namely those with no free variables; an expression with free variables can be thought of as an agent schema. In a later section, we find a natural congruence relation over the language; when we quotient the language by this congruence, we immediately obtain a mathematical interpretation, and a rich set of algebraic identities which—together with the derivation rules—justifies the term 'calculus of agents'.

We presuppose an infinite set $\mathrm{Var} = \{X, X_1, \ldots, Y, Y_1, \ldots\}$ of *agent variables*. Apart from these, our expressions are formed by four operators: Action, Summation, Product and Restriction. To admit agents with infinite behaviour, i.e. infinite derivation sequences, we also add a recursion construct. Finally, for ease of expression we include a fifth operator based upon morphisms of the Action monoid, though later we shall show that under a natural assumption it can be defined in terms of the other operators plus recursion.

We shall use E, F, G, possibly indexed, to stand for arbitrary expressions. The notations and interpretations of the four operators (or, more strictly, operator classes) are as follows:

(1) *Action*: $a:E$ is an expression, for each $a \in \mathrm{Act}$. It has just one action:

$$a:E \xrightarrow{a} E \quad .$$

Thus the unary operator '$a:$' has the effect of prefixing the atomic action a to the behaviour represented by E.

(2) *Summation*: $\sum \tilde{E}$ is an expression, where $\tilde{E} = \langle E_i \mid i \in I \rangle$ is some indexed family of expressions (possibly infinite). We write also $\sum_{i \in I} E_i$. Its derivation rule is

$$\frac{E_i \xrightarrow{a} E}{\sum_{i \in I} E_i \xrightarrow{a} E} \quad (i \in I) \quad .$$

That is, the action below the line may be inferred from any action $E_i \xrightarrow{a} E$, for some $i \in I$.

The special cases $I = 0$ and $I = 2$ are important—indeed, for finitely expressed agents it turns out that they are all we need. We write

$$\mathbf{0} \quad \text{for } \sum_{i \in 0} E_i, \qquad E_0 + E_1 \quad \text{for } \sum_{i \in 2} E_i.$$

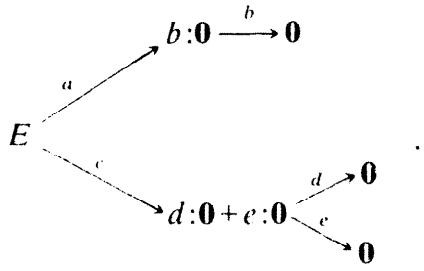
We may call **0** *inaction*; it has no actions, since none can be inferred. The rules for $I = 2$ can be written

$$\boxed{\frac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'}} \qquad \boxed{\frac{F \xrightarrow{a} F'}{E+F \xrightarrow{a} F'}} .$$

Summation represents the superposition of agents. Each action of a sum determines one summand as the source of future behaviour. Consider for example

$$E \equiv a:b:\mathbf{0}+c:(d:\mathbf{0}+e:\mathbf{0})$$

(we assume sum to have lower binding power than action); its derivatives and their actions, as inferred by the above rules, can be collected into a *derivation tree*:



A sum may represent non-determinism; if $a = c$ in the above example then the action a may have one of two possible outcomes. On the other hand we shall see later how, with the help of the restriction operator, other factors of a product containing E can determine E's course of action.

(3) *Product*: $E \times F$ is an expression. We have already discussed the intuition of product; its derivation rule is

$$\boxed{\frac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E \times F \xrightarrow{ab} E' \times F'}} .$$

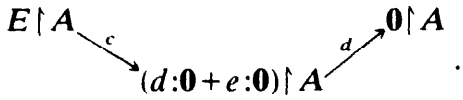
Note that we do not permit infinitary product, in contrast to sum. If we did so, the derivation rule would have an infinite set of hypotheses, and we would require infinite product in the Action monoid. Apart from these technical complications, we would also be allowing a single atomic action to accomplish an infinite amount of computation, and there is little point in departing from reality to this extent. An infinite *sum*, on the other hand, allows a single atomic action to discard an infinite set of alternatives; this is apparently more realistic and leads to few technical difficulties.

(4) *Restriction*: $E \restriction A$ is an expression, for any subset A of Act containing 1. Its derivation rule is

$$\boxed{\frac{E \xrightarrow{a} E'}{E \restriction A \xrightarrow{a} E' \restriction A} \quad (a \in A)} \ .$$

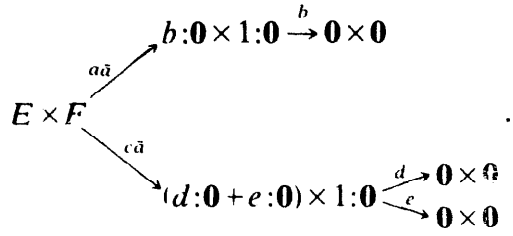
Thus the effect of $\restriction A$ is to inhibit any action which is not in A.

As a first example, consider $E \restriction A$ where E is the example discussed under sum above, and $A = \{1, c, d\}$. The derivation tree for $E \restriction A$ is

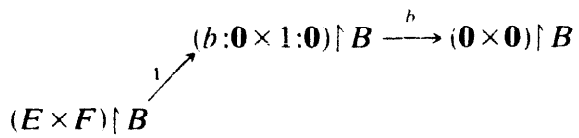


It has been pruned of all actions not in A.

But the derivation of E can also be directed by another factor in a restricted product. Suppose that $\bar{a}$ is an inverse of a, i.e. $a\bar{a} = 1$, and let $F \equiv \bar{a}:1:\mathbf{0}$. The derivation tree of $E \times F$ looks like this:



If we choose $B = \{1, b, d, e\}$ then, remembering $a\bar{a} = 1$, the restricted product $(E \times F) \restriction B$ discards the lower branch (assuming $c\bar{a} \notin B$):



Moreover, the *upper* branch would be discarded instead by taking $F \equiv \bar{c}:1:\mathbf{0}$. More generally, in a larger expression F may adopt either of these forms – depending upon how earlier derivations were similarly directed (perhaps by an external observer or experimenter).

It should be noted that '$\restriction A$', for each A, is a unary operator upon agents. We do not care what syntactic forms are used for presenting such subsets A of the action space.

The four operators allow complex agents to be defined, but they will have only finite derivations. To model persistent agents we admit now a form of recursion.

Recursion. $\mathrm{fix}_i \tilde{X}\tilde{E}$ is an expression, where $\tilde{E} = \langle E_i \mid i \in I \rangle$ is an I-indexed family of expressions, $\tilde{X} = \langle X_i \mid i \in I \rangle$ is an I-indexed family of distinct variables and $i \in I$.

Intuitively $\mathrm{fix}_i \tilde{X}\tilde{E}$ stands for the ith component of a 'solution' of the family of equations $\tilde{X} = \tilde{E}$, but 'equality' between expressions is not yet explained. The unsubscripted form $\mathrm{fix}\, \tilde{X}\tilde{E}$ stands for the family $\langle \mathrm{fix}_i \tilde{X}\tilde{E} \mid i \in I \rangle$.

The prefix $\mathrm{fix}_i\tilde{X}$ binds each variable X_j. Having now listed all expression forms, we understand the concepts of free and bound variables in the usual way, and define substitution.

Definition. We denote by Free(E), Free($\tilde{E}$) the sets of variables free in E, $\tilde{E}$. We denote by $E\{\tilde{G}/\tilde{X}\}$ the result of simultaneously replacing G_i for free occurrences of X_i in E, for each i, with change of bound variables as necessary to avoid clashes.

The derivation rule for recursion is as follows:

$$\frac{E_i\{\mathrm{fix}\,\tilde{X}\tilde{E}/\tilde{X}\} \xrightarrow{a} E'}{\mathrm{fix}_i\,\tilde{X}\tilde{E} \xrightarrow{a} E'}.$$

Clearly the special case $I = 1$ is of interest. In fact, for finitely expressed agents—where I is restricted to be finite—it can be shown to be all that we need, though we shall not prove this fact.

We now give some simple examples of recursion.

(i) $F \equiv \mathrm{fix}\,XX$. The only instances of the rule which can yield an action for F have the form

$$\frac{F \xrightarrow{a} E'}{F \xrightarrow{a} E'}.$$

Hence, since every derivation must be inferred by a finite proof, F in this case has no actions. It will turn out to be congruent to $\mathbf{0}$.

(ii) $F \equiv \mathrm{fix}\,X(1{:}X)$. The only action for F is given by the proof

$$\frac{1{:}F \xrightarrow{1} F}{F \xrightarrow{1} F}$$

where the action above the line follows from the rule for Action (1). So the derivation tree for F is

$$F \xrightarrow{1} F \xrightarrow{1} F \xrightarrow{1} \cdots.$$

This leads us to define $\mathbf{1} \equiv \mathrm{fix}\,X(1{:}X)$; it will turn out that $\mathbf{1}$ is an identity for product ($\times$), up to congruence.

(iii) More generally, we may define the unary operator δ (delay) by

$$\delta E \equiv \mathrm{fix}\,X(1{:}X + E) \quad (X \text{ not free in } E).$$

Then the following *derived* rules are easily seen to yield exactly the actions of δE:

$$\boxed{\delta E \xrightarrow{1} \delta E} \qquad \boxed{\frac{E \xrightarrow{a} E'}{\delta E \xrightarrow{a} E'}}.$$

Intuitively, δE may idle indefinitely before behaving like E.

(iv) $F_i \equiv \mathrm{fix}_i\langle X_0, X_1\rangle\langle a:X_0+b:X_1, c:X_0+d:X_1\rangle$, $i=0, 1$. The rules for F_0 and F_1 are then

$$\frac{a:F_0+b:F_1 \xrightarrow{f} F'}{F_0 \xrightarrow{f} F'}, \qquad \frac{c:F_0+d:F_1 \xrightarrow{f} F'}{F_1 \xrightarrow{f} F'}.$$

But this just says that F_0 acts like $a:F_0+b:F_1$ and F_1 acts like $c:F_0+d:F_1$; for this reason we can often avoid formal use of recursion expressions by introducing constant symbols by definition. In general, a set of mutually recursive definitions of constants K_i

$$K_i \Leftarrow E_i \quad (i \in I)$$

can be understood as a set of direct definitions

$$K_i \equiv \mathrm{fix}_i\hat{X}\langle\hat{E}\{\hat{X}/\hat{K}\}\rangle$$

and hence, for each constant K_i, we immediately have the rule

$$\frac{E_i \xrightarrow{a} E'}{K_i \xrightarrow{a} E'}.$$

This completes the basic constructs of SCCS; the syntax and rules are collected together in Table 1. An *agent* is taken to be an expression with no free variables, and we shall use $\mathscr{E}$, $\mathscr{P}$ to stand for the classes of expressions and agents respectively.

It remains to specify the morphism operator, which is in fact derivable under further natural assumptions.

Morphism: $E[\phi]$ is an expression, where $\phi: \mathrm{Act} \to \mathrm{Act}$ is a monoid morphism. Its derivation rule is

$$\boxed{\frac{E \xrightarrow{a} E'}{E[\phi] \xrightarrow{\phi(a)} E'[\phi]}}.$$

Table 1
Expressions and derivation rules in SCCS.

Syntax $E ::= X \mid a:E \mid \sum \tilde{E} \mid E \times E \mid E \restriction A \mid \mathrm{fix}_i \tilde{X}\tilde{E}$

Rules

Action $\boxed{a:E \xrightarrow{a} E}$ Sum: $\boxed{\dfrac{E_i \xrightarrow{a} E'}{\sum \tilde{E} \xrightarrow{a} E'}}$

Product: $\boxed{\dfrac{E \xrightarrow{a} E' \quad F \xrightarrow{b} F'}{E \times F \xrightarrow{ab} E' \times F'}}$ Restriction: $\boxed{\dfrac{E \xrightarrow{a} E'}{E \restriction A \xrightarrow{a} E' \restriction A}\ (a \in A)}$

Recursion: $\boxed{\dfrac{E_i\{\mathrm{fix}\, \tilde{X}\tilde{E}/\tilde{X}\} \xrightarrow{a} E'}{\mathrm{fix}_i \tilde{X}\tilde{E} \xrightarrow{a} E'}}$

Derived operators $E ::= \cdots \mid \mathbf{0} \mid \delta E \mid E[\phi]$

Rules

Delay: $\boxed{\delta E \xrightarrow{1} \delta E}$ Morphism: $\boxed{\dfrac{E \xrightarrow{a} E'}{E[\phi] \xrightarrow{\phi(a)} E'[\phi]}}$

$\boxed{\dfrac{E \xrightarrow{a} E'}{\delta E \xrightarrow{a} E'}}$

Thus $[\phi]$ has the simple effect of applying ϕ to the actions of all derivatives of E. It is useful for generating copies of an agent, differing only in the values of their actions. For example if $P \equiv \mathrm{fix}\, X(a:b:X)$, and $\phi(a) = c$, $\phi(b) = d$, then $P[\phi]$ is (up to congruence) the agent $\mathrm{fix}\, X(c:d:X)$.

Further assumptions: In discussing restriction we found it useful to have an inverse $\bar{a}$ for an action a. The existence of inverses is so convenient that, although many properties of the calculus do not require them, we shall further postulate that

$$\boxed{(\mathrm{Act}, \times, 1, \bar{\ }) \text{ is an Abelian Group}}\ .$$

The effect is that any action a may be observed by (or used as a vehicle for communication with) an agent who is capable of the inverse action $\bar{a}$.

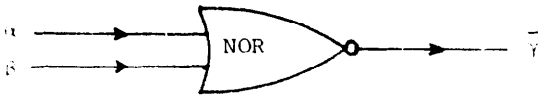
For technical purposes it is convenient to assume that no agent explores the whole space of possible actions. Put in another way, we shall allow that the action group Act can always be extended to $\mathrm{Act} \otimes \mathrm{Act}'$ (direct product of Abelian groups) where Act' is again an Abelian group.

Our first use of these assumptions is to show that the morphism operator $[\phi]$ is definable. (Note in passing that any morphism $\phi: A \to B$ of monoids is also a group morphism when A and B are groups.) The proof is given in the Appendix. The importance of this result is that with only four basic operations we can claim to be closer to an irreducible set, and also the case analysis in proofs about the calculus is not too tedious.

3. Examples: idealised switching circuits and 'timeout'

In our first example we show how the behaviour of a simple switching circuit can be modelled in the calculus in a somewhat idealised fashion. This will illustrate the use of all our operators, and provide a simple proof study using algebraic laws which we develop later. Moreover, it will show how it is convenient to assume an action monoid which is also an Abelian group; the inverse operation in this group gives us exactly what we need for composing circuits from subcircuits.

Consider a switching element



We interpret it as follows: if boolean values $i, j \in \{0, 1\}$ are input at 'ports' α, β at time t, then the boolean value $i \downarrow j$ will be output at $\bar{\gamma}$ at time $t+1$. (The 'nor' function $i \downarrow j$ is given by $\neg i \wedge \neg j$.) Thus the element is capable of performing, at any time instant, certain combinations of the six actions α_0, α_1, β_0, β_1, $\bar{\gamma}_0$ and $\bar{\gamma}_1$.

It is convenient for this example, and perhaps in general, to take the action group Act to be the free Abelian group generated by a denumerably infinite set Σ of *names*; typical members of Σ will here be α_i, β_i, γ_i, ... for $i \in \{0, 1\}$. The *conames* $\bar{\Sigma} = \{\bar{\sigma} \mid \sigma \in \Sigma\}$ together with the names Σ may be called the *particulate* actions; since Act is freely generated by Σ, any action can be expressed uniquely, up to order, as a finite product

$$\sigma_1^{z_1}\sigma_2^{z_2} \cdots \quad (z_i \neq 0)$$

of powers of names. Note the usual convention that $\sigma^{-n} = \bar{\sigma}^n$; also, 1 is identified with the empty product. We will write Λ for $\Sigma \cup \bar{\Sigma}$.

Returning to the NOR element, since it has a unit delay it has a one-bit memory capacity. Denoting by NOR(k) the element with current output k, the behaviour of the element can be defined in the calculus by

$$\boxed{\mathrm{NOR}(k) \Leftarrow \sum_{i,j \in \{0,1\}} (\alpha_i \beta_j \bar{\gamma}_k : \mathrm{NOR}(i \downarrow j))} \quad . \tag{1}$$

This is a definition of two agents NOR(0), NOR(1) by mutual recursion. Note that the product $\alpha_i \beta_j \bar{\gamma}_k$ is a single atomic (but non-particulate) action.

Although not necessary, for our example we have chosen to use names for 'input' and conames for 'output'. As we shall see, the inverse operator is mainly significant not for this connotation (which is merely convenient) but for the purpose of joining elements together.

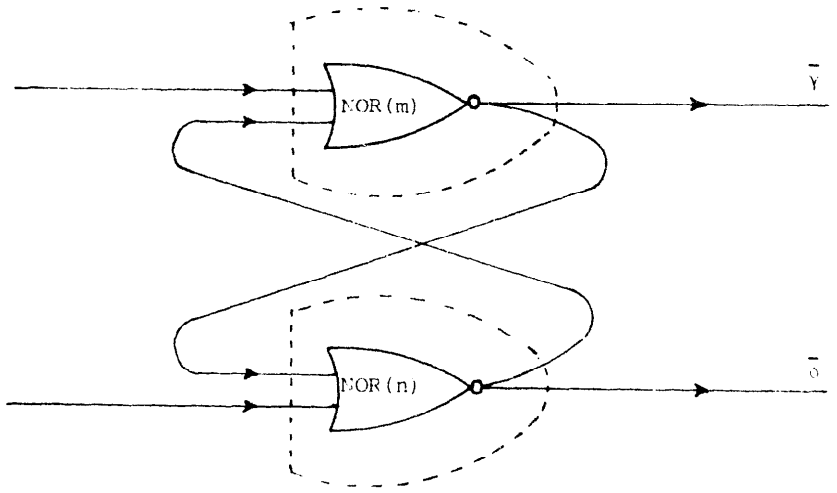
Before going further, we may also remark that a useful convention, for avoiding excessive appearance of the summation symbol, is to use variables x, y, ... over a

suitable data domain (here {0, 1}) to imply summation. Then our definition may be written as

$$\mathrm{NOR}(k) \Leftarrow \alpha_x \beta_y \bar{\gamma}_k : \mathrm{NOR}(x \downarrow y).$$

The first occurrences of x and y may be regarded as *variable binding* occurrences. However, we shall not have enough need for this convention in the present paper to justify its use, except briefly in Section 6 in relation to CCS.

Two NOR elements may be composed in a familar way to make a Set–Reset Flip-flop (with a two-bit memory capacity):

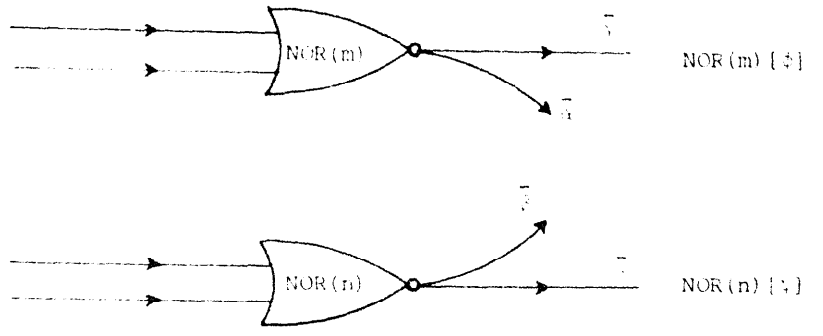


To achieve the composition, we first have to use morphisms to make two appropriately relabelled copies of the NOR element, each with two particulate actions instead of one for output, since the output lines fork into two. We choose the two morphisms

ϕ under which $\alpha_i \mapsto \sigma_i$, $\gamma_i \mapsto \gamma_i \alpha_i$,

ψ under which $\beta_i \mapsto \rho_i$, $\gamma_i \mapsto \beta_i \delta_i$

which leave all other names unchanged (any morphism is determined by its values on the generators). They may be written respectively $\phi = \sigma/\alpha$, $\gamma\alpha/\gamma$ and $\psi = \rho/\beta$, $\beta\delta/\gamma$. The resulting agents may be diagrammed by



Now that lines to be joined have been inversely named, we merely have to form the product of these two agents, and to represent the 'internalisation' of lines $\bar{\alpha} \to \alpha$ and $\bar{\beta} \to \beta$ by restriction. For this restriction $\restriction A$, we could choose A to be the submonoid of Act generated by $\{\sigma_i, \rho_i, \gamma_i, \delta_i\}$, but it is more useful in general to define the restriction operator $\backslash N$, for any set N of particles, as follows:

$$E\backslash N = E \restriction (\mathcal{A} - N)^*$$

where L^*, for $L \subseteq \mathcal{A}$, is the submonoid of Act generated by L. Thus, abbreviating the set $\{\alpha_0, \alpha_1\}$ by α, the Flip-flop is just

$$\boxed{\mathrm{FF}(m, n) \equiv (\mathrm{Nor}(m)[\phi] \times \mathrm{Nor}(n)[\psi])\backslash\alpha\backslash\beta} \quad . \tag{2}$$

This example illustrates how an input port α, carrying data from a data domain D, may be represented by a subset $\{\alpha_d \mid d \in D\}$ of Σ, and similarly for output ports. It also shows how appropriate combinations of morphism, product and restriction may be used to model the channelling of data, along what may be thought of as wires.

But we have so far only tried to persuade the reader by diagrams that FF, so constructed, behaves properly. We shall now anticipate the results of later sections to *prove* that it does. The reader is recommended to read this proof, and that of the next example, cursorily at first; the later sections will justify the formal manipulations.

In automata-theoretic terms the Flip-flop has four states, represented by a pair of boolean values (m, n). Moreover in a typical state:

(i) The current outputs at $\bar{\gamma}$, $\bar{\delta}$ are m, n;

(ii) The next state is $(i \downarrow n, m \downarrow j)$, where i, j are the current inputs.

In our terms, this means that we wish to prove the equation

$$\mathrm{FF}(m, n) \sim \sum_{i,j \in \{0,1\}} \sigma_i \rho_j \bar{\gamma}_m \bar{\delta}_n : \mathrm{FF}(i \downarrow n, m \downarrow j)$$

where '$\sim$' (the congruence of Section 4) stands for congruent behaviour. To establish this, we first calculate

$$\mathrm{NOR}(m)[\phi] \sim \sum_{i,h} \sigma_i \beta_h \bar{\gamma}_m \bar{\alpha}_m : (\mathrm{NOR}(i \downarrow h)[\phi])$$

using (1) with Propositions 4.7, 5.6(1), 5.6(2), and similarly

$$\mathrm{NOR}(n)[\psi] \sim \sum_{k,j} \alpha_k \rho_j \bar{\beta}_n \bar{\delta}_n : (\mathrm{NOR}(k \downarrow j)[\psi]).$$

Now the product of these two terms, by the distributive law, Proposition 5.1(3), is the sum of sixteen terms indexed by i, h, k, j; the double restriction $\backslash\alpha\backslash\beta$ reduces to $\mathbf{0}$ all summands except those in which $h = n$ and $k = m$, by Proposition 5.4(1),

and from the definition (2) we are left with

$$\mathrm{FF}(m,n) \sim \sum_{i,j} \sigma_i \rho_j \bar{\gamma}_m \bar{\delta}_n : (\mathrm{NOR}(i \downarrow n)[\phi] \times \mathrm{NOR}(m \downarrow j)[\psi]) \backslash \alpha \backslash \beta$$

$$\equiv \sum_{i,j} \sigma_i \rho_j \bar{\gamma}_m \bar{\delta}_n : \mathrm{FF}(i \downarrow n, m \downarrow j) \quad \text{as required.}$$

Moreover, it follows from Proposition 4.9 that this equation has a unique solution, up to congruence. In fact the solution has a fixed point notation in the calculus; if we choose $I = \{0,1\}^2$ and take $\tilde{X}$ to be an I-indexed family of variables $X(m,n)$, then

$$\mathrm{FF}(m,n) \sim \mathrm{fix}_{m,n} \tilde{X} \tilde{E}, \quad \text{where } E_{m,n} \equiv \sum_{i,j} \sigma_i \rho_j \bar{\gamma}_m \bar{\delta}_n : X(i \downarrow n, m \downarrow j).$$

For our second example we give a simple case of what is known as 'timeout'. Suppose that P may or may not perform an action σ at some future instant; for simplicity we suppose that P is one of

$$P_n \equiv \cdot(1:)^n \sigma : \mathbf{1}, \quad n \in \omega, \qquad P_\omega \equiv \mathbf{1}.$$

Suppose also that either agent Q or agent R is to be initiated, according as P does or does not perform σ before the kth time instant, i.e. $n < k$. We assume that neither Q nor R ever performs σ or $\bar{\sigma}$; in terms of Section 5, this means that neither σ nor $\bar{\sigma}$ is in the *sort* of Q or R.

Then for each $k \in \omega$, we would like to define some context or environment $\mathscr{C}_k[\ ]$ for P such that for $n \leqslant \omega$

$$\mathscr{C}_k[P_n] \sim \begin{cases} (1:)^{n+1} Q & \text{if } n < k \ (P \text{ does } \sigma \text{ in time}), \\ (1:)^k R & \text{if } n \geqslant k \ (\text{'timeout'}). \end{cases}$$

We claim that this is achieved by defining

$$\mathscr{C}_k[P] \equiv (P \times W_k) \backslash \{\sigma, \bar{\sigma}\}$$

where W_k ('wait up to k') is given by

$$W_0 \equiv R \times \delta(\bar{\sigma}:\mathbf{1}), \qquad W_{k+1} \equiv \bar{\sigma}:Q + 1:W_k.$$

The second factor of W_0 is just to absorb any 'late' σ performed by P.

Now at the risk of boring the reader, we prove our claim in complete detail, annotating each proof step with the equational laws which justify it. Once our algebra is understood such manipulations may be greatly abbreviated; for the present, we are concerned to show that we do indeed have enough laws, and that the manipulations are not too unfamiliar, since we are working in a commutative semi-ring.

Let us write $\backslash\{\sigma, \bar{\sigma}\}$ as $\backslash\backslash\sigma$. Then in the simple case $P \equiv P_\omega$ we have

$$\begin{aligned}\mathscr{C}_k[P_\omega] &\equiv (\mathbf{1} \times W_k)\backslash\backslash\sigma \\ &\sim W_k\backslash\backslash\sigma \quad \text{by Proposition 5.1(1).}\end{aligned}$$

But by easy induction on k, we can show $W_k\backslash\backslash\sigma \sim (1:)^k R$:

$$\begin{aligned}W_0\backslash\backslash\sigma &\sim (R \times \delta(\bar{\sigma}:\mathbf{1}))\backslash\backslash\sigma \\ &\sim R\backslash\backslash\sigma \times \delta(\bar{\sigma}:\mathbf{1})\backslash\backslash\sigma \quad \text{by Proposition 5.9(3)} \\ &\sim R \times \delta\mathbf{0} \quad \text{by Propositions 5.9(1), 5.5(2), 5.4(1)} \\ &\sim R \quad \text{by Proposition 5.1(1').}\end{aligned}$$

and

$$\begin{aligned}W_{k+1}\backslash\backslash\sigma &\sim (\bar{\sigma}:Q + 1:W_k)\backslash\backslash\sigma \\ &\sim (\bar{\sigma}:Q)\backslash\backslash\sigma + 1:(W_k\backslash\backslash\sigma) \quad \text{by Propositions 5.4(2), 5.4(1)} \\ &\sim \mathbf{0} + 1:(1:)^k R \quad \text{by Proposition 5.4(1) and induction} \\ &\sim (1:)^{k+1} R \quad \text{by Proposition 5.1(2).}\end{aligned}$$

For the case $P \equiv P_n$, $n \in \omega$, we first prove a reduction:

$$\begin{aligned}\mathscr{C}_{k+1}[P_{n+1}] &\equiv (1:P_n \times (\bar{\sigma}:Q + 1:W_k))\backslash\backslash\sigma \\ &\sim (\bar{\sigma}:(P_n \times Q) + 1:(P_n \times W_k))\backslash\backslash\sigma \quad \text{by Propositions 5.1(3), 5.3} \\ &\sim \mathbf{0} + 1:((P_n \times W_k)\backslash\backslash\sigma) \text{ by Propositions 5.4(2), 5.4(1)} \\ &\sim 1:\mathscr{C}_k[P_n].\end{aligned}$$

The main claim now follows by considering the two cases $n = 0 < k$ and $n \geq 0 = k$:

$$\begin{aligned}\mathscr{C}_{k+1}[P_0] &\equiv (\sigma:\mathbf{1} \times (\bar{\sigma}:Q + 1:W_k))\backslash\backslash\sigma \\ &\sim (1:Q + \sigma:W_k)\backslash\backslash\sigma \quad \text{by Propositions 5.1(3), 5.3, 5.1(1)} \\ &\sim 1:Q \quad \text{by Propositions 5.4(2), 5.4(1), 5.9(1),} \\ \mathscr{C}_0[P_n] &\equiv (P_n \times (R \times \delta(\bar{\sigma}:\mathbf{1})))\backslash\backslash\sigma \\ &\sim R\backslash\backslash\sigma \times (P_n \times \delta(\bar{\sigma}:\mathbf{1}))\backslash\backslash\sigma \quad \text{by Propositions 5.1(1), 5.9(3)} \\ &\sim R \quad \text{by Propositions 5.9(1), 5.1(1) and below.}\end{aligned}$$

The last step depends on the inductive proof that $(P_n \times \delta(\bar{\sigma}:\mathbf{1}))\backslash\backslash\sigma \sim \mathbf{1}$:

$$\begin{aligned}(P_0 \times \delta(\bar{\sigma}:\mathbf{1}))\backslash\backslash\sigma &\sim (\sigma:\mathbf{1} \times (\bar{\sigma}:\mathbf{1} + 1:\delta(\bar{\sigma}:\mathbf{1})))\backslash\backslash\sigma \quad \text{by Proposition 5.5(4)} \\ &\sim (1:\mathbf{1} + \sigma:\delta(\bar{\sigma}:\mathbf{1}))\backslash\backslash\sigma \quad \text{by Propositions 5.1(3), 5.3} \\ &\sim 1:\mathbf{1} \quad \text{by Propositions 5.4(2), 5.9(1), 5.4(1),} \\ &\sim \mathbf{1} \quad \text{by Propositions 5.1(2), 5.4(4) since } \mathbf{1} \equiv \delta 0.\end{aligned}$$

and

$$\begin{aligned}(P_{n+1}\times\delta(\bar{\sigma}\cdot\mathbf{1}))\backslash\backslash\sigma &\sim (1{:}P_n\times(\bar{\sigma}{:}\mathbf{1}+1{:}\delta(\bar{\sigma}{:}\mathbf{1})))\backslash\backslash\sigma \quad \text{by Proposition 5.5(4)}\\ &\sim (\bar{\sigma}{:}P_n+1{:}(P_n\times\delta(\bar{\sigma}{:}\mathbf{1})))\backslash\backslash\sigma\\ &\qquad \text{by Propositions 5.1(3), 5.3}\\ &\sim 1{:}(P_n\times\delta(\bar{\sigma}{:}\mathbf{1}))\backslash\backslash\sigma \quad \text{by Propositions 5.4(2), 5.4(1)}\\ &\sim 1{:}\mathbf{1} \quad \text{by induction}\\ &\sim \mathbf{1}.\end{aligned}$$

With the details of the proof all visible, we may isolate some characteristics.

(i) The distributive law, 5.1(3), for product over sum is quite important. In CCS [8] this law is absent.

(ii) The distribution of restriction over sum, 5.4(2), is always valid.

(iii) The distribution of restriction over product, 5.9(3), is valid under certain limitations (on sorts), and its use is critical.

4. Strong bisimulation and congruence

We do not wish to distinguish agents which have, in some sense, the same derivation tree. This is made precise by the notion of *bisimulation* between agents.

Definition. A binary relation $\mathcal{R}\subseteq\mathcal{P}^2$ is a *strong bisimulation* if, whenever $P\mathcal{R}Q$ and $a\in\mathrm{Act}$,

(i) if $P\xrightarrow{a}P'$ then, for some Q', $Q\xrightarrow{a}Q'$ and $P'\mathcal{R}Q'$,

(ii) if $Q\xrightarrow{a}Q'$ then, for some P', $P\xrightarrow{a}P'$ and $P'\mathcal{R}Q'$.

Diagrams make the idea clear:

(i)
$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle a}\downarrow & & \\ P' & & \end{array} \quad \text{implies, for some } Q' \quad \begin{array}{c} Q \\ {\scriptstyle a}\downarrow \\ P'\ \mathcal{R}\ Q' \end{array}$$

(ii)
$$\begin{array}{ccc} P & \mathcal{R} & Q \\ & & {\scriptstyle a}\downarrow \\ & & Q' \end{array} \quad \text{implies, for some } P' \quad \begin{array}{c} P \\ {\scriptstyle a}\downarrow \\ P'\ \mathcal{R}\ Q'. \end{array}$$

(In Part II we consider other bisimulations for different action relations in place of $\xrightarrow{a}$.)

Now for any $\mathcal{R}\subseteq\mathcal{P}^2$ we may define $\mathcal{F}(\mathcal{R})$ to be the set of pairs $\langle P, Q\rangle$ satisfying clauses (i) and (ii). In terms of this map $\mathcal{F}$ of relations, it is clear that $\mathcal{R}$ is a strong bisimulation iff $\mathcal{R}\subseteq\mathcal{F}(\mathcal{R})$.

Proposition 4.1. *$\mathcal{F}$ is monotonic over the lattice of binary relations under inclusion.*

The following is then standard. (For the rest of this section we omit the adjective 'strong' qualifying 'bisimulation'.)

Proposition 4.2. *There exists a maximum bisimulation given by* $\sim = \bigcup\{\mathcal{R} \mid \mathcal{R} \subseteq \mathcal{F}(\mathcal{R})\}$. *Moreover* $\sim = \mathcal{F}(\sim)$, *and indeed '$\sim$' is the maximum fixed-point of $\mathcal{F}$.*

Proposition 4.3. *'$\sim$' is an equivalence relation.*

Proof. Note that $P \sim Q$ iff, for some bisimulation $\mathcal{R}$, $P\mathcal{R}Q$. Then the result follows from the fact that the identity $\mathrm{Id}_{\mathcal{P}}$ is a bisimulation, and that the composition and converse of bisimulations are bisimulations. □

In CCS [8] a notion of strong congruence ($\sim$) was given which could equally well have been defined by the present method. It was argued there that this is an intuitively correct equivalence relation to adopt, though for some purposes one may be satisfied with a larger (or coarser) relation, and we shall not repeat the argument here. However, the present method of definition in terms of bisimulations, which is due to Park [15], has distinct advantages. First, the method of [8] required a constraint upon the recursive definitions allowed, and would also run into difficulty with the infinitary language introduced here.

Second, the present method admits an elegant proof technique; to show $P \sim Q$, it is necessary and sufficient to find a bisimulation containing the pair $\langle P, Q\rangle$. Our first uses of this technique are in proving '$\sim$' to be a congruence.

First, we note that change of bound variables in agents respects bisimulation. If we define $E \equiv F$ to mean that E and F are identical up to change of bound variables, then we have the following basic proposition.

Proposition 4.4. *$P \equiv Q$ implies $P \sim Q$.*

Proof. It is enough to show that '$\equiv$' is itself a bisimulation. The details are routine, using the rules of action, and we omit them. We merely note that the substitution operation respects '$\equiv$'; that is,

$$E \equiv F \text{ and } \tilde{P} \equiv \tilde{Q} \quad \text{imply} \quad E\{\tilde{P}/\tilde{X}\} \equiv F\{\tilde{Q}/\tilde{X}\}.$$

This property is needed in the proof, for actions inferred by the recursion rule. □

It is this fact which allows us, in the sequel, to treat '$\equiv$' as we would treat syntactic identity.

It is very convenient in proofs to use the notion of bisimulation up to '$\sim$', which we now make precise. We follow normal practice in taking the composition of

$\mathcal{R}, \mathcal{S} \subseteq \mathcal{P}^2$ to be

$$\mathcal{R}\mathcal{S} = \{\langle P_1, P_3\rangle \mid \text{for some } P_2,\ P_1 \mathcal{R} P_2 \text{ and } P_2 \mathcal{S} P_3\}$$

and we are particularly interested in the composition $\sim\mathcal{R}\sim$.

Definition. $\mathcal{R}$ is a *bisimulation up to* '$\sim$' iff $\sim\mathcal{R}\sim$ is a bisimulation.

Proposition 4.5. *$\mathcal{R}$ is a bisimulation up to '$\sim$' iff $\mathcal{R} \subseteq \mathcal{F}(\sim\mathcal{R}\sim)$.*

Proof. Routine, using the general propety $\mathcal{F}(\mathcal{R})\mathcal{F}(\mathcal{S}) \subseteq \mathcal{F}(\mathcal{R}\mathcal{S})$. □

The usefulness of this proposition will appear in Proposition 4.6.

Before tackling the proof of congruence, we extend the relation '$\sim$' from agents $\mathcal{P}$ to expressions $\mathcal{E}$ in the natural way.

Definition. Let $\tilde{X}$ include the free variables of E and F. Then $E \sim F$ if $E\{\tilde{P}/\tilde{X}\} \sim F\{\tilde{P}/\tilde{X}\}$ for all agents $\tilde{P}$.

Proposition 4.6. *The relation '$\sim$' is a congruence; that is,*

(1) *$E \sim F$ implies $a{:}E \sim a{:}F$, $E \times G \sim F \times G$ and $E \restriction A \sim F \restriction A$;*

(2) *$\tilde{E} \sim \tilde{F}$ implies $\sum \tilde{E} \sim \sum \tilde{F}$ and $\mathrm{fix}_i \tilde{X}\tilde{E} \sim \mathrm{fix}_i \tilde{X}\tilde{F}$.*

Proof. (1) It is enough to do the proof only for agents, since the extension to expressions is routine by the previous definition; we shall just prove one case:

$$P \sim Q \quad \text{implies} \quad P \times R \sim Q \times R.$$

For this it is enough to show that $\mathcal{R} = \{\langle P \times R, Q \times R\rangle \mid P \sim Q, R \in \mathcal{P}\}$ is a bisimulation. Suppose $(P \times R)\mathcal{R}(Q \times R)$ and $R \times R \xrightarrow{c} P' \times R'$ (the successor must be of this form). Then $P \xrightarrow{a} P'$, $R \xrightarrow{b} R'$ and $ab = c$. But then, since $P \sim Q$, $Q \xrightarrow{a} Q'$ for some $Q' \sim P'$, hence $Q \times R \xrightarrow{c} Q' \times R'$ and $(P' \times R')\mathcal{R}(Q' \times R')$. By a symmetric argument, $\mathcal{R}$ is therefore a bisimulation.

(2) For summation, it is enough to show that

$$\{\langle \sum \tilde{P}, \sum \tilde{Q}\rangle \mid \tilde{P} \sim \tilde{Q}\} \cup \mathrm{Id}_{\mathcal{P}}$$

is a bisimulation. We omit the details. For recursion, for the sake of clarity we shall just prove that when at most X is free in E or F, so that $\mathrm{fix}\, XE$ and $\mathrm{fix}\, XF$ are agents, then

$$E \sim F \quad \text{implies} \quad \mathrm{fix}\, XE \sim \mathrm{fix}\, XF.$$

The extension to mutual recursion is only notationally more cumbersome, and the case where there are other free variables is routine by the preceding definition. Assuming $E \sim F$, we shall show that

$$\mathcal{R} = \{\langle G\{\mathrm{fix}\, XE/X\}, G\{\mathrm{fix}\, XF/X\}\rangle \mid \mathrm{Free}(G) \subseteq \{X\}\}$$

is a bisimulation up to '$\sim$'; the result follows by taking $G \equiv X$. So we shall show by induction on inference that if

$$G\{\text{fix } XE/X\} \xrightarrow{a} P', \text{ then for some } Q'$$

$$G\{\text{fix } XF/X\} \xrightarrow{a} Q' \text{ and } P' \mathcal{R} \sim Q'. \qquad (\bigstar)$$

Consider the possible forms of G; we omit some forms where the proof is analogous to those treated.

(i) $G \equiv X$. Then fix $XE \xrightarrow{a} P'$, so by a shorter inference $E\{\text{fix } XE/X\} \xrightarrow{a} P'$, and by induction $E\{\text{fix } XF/X\} \xrightarrow{a} Q'$ with $P' \mathcal{R} \sim Q'$. But since $E \sim F$, by definition $E\{\text{fix } XF/X\} \sim F\{\text{fix } XF/X\}$, so $F\{\text{fix } XF/X\} \xrightarrow{a} Q'' \sim Q'$ and $P' \mathcal{R} \sim Q''$. But then by the recursion rule $G\{\text{fix } XF/X\} \equiv \text{fix } XF \xrightarrow{a} Q''$ also, and we are done.

(ii) $G = G_0 \times G_1$. Then $G_i\{\text{fix } XE/C\} \xrightarrow{a_i} P_i'$, $i \in \{0, 1\}$, with $a_0 a_1 = a$ and $P_0' \times P_1' \equiv P'$. Hence by induction $G_i\{\text{fix } XF/X\} \xrightarrow{a_i} Q_i'$, with $P_i' \sim Q_i'$, $i \in \{0, 1\}$). Thus $G\{\text{fix } XF/X\} \xrightarrow{a} Q' \equiv Q_0' \times Q_1'$, and by a simple argument together with part (1) of the proposition we can show that $P' \sim Q'$ as required.

(iii) $G \equiv \text{fix } YH$, $Y \not\equiv X$. Then we have, by assumption, that fix $Y(H\{\text{fix } XE/X\}) \xrightarrow{a} P'$, so by a shorter inference we have $H\{\text{fix } XE/X\}\{G\{\text{fix } XE/X\}/Y\} \xrightarrow{a} P'$, which may be rewritten as $H\{G/Y\}\{\text{fix } XE/X\} \xrightarrow{a} P'$. So by induction, applied to the expression $H\{G/Y\}$, we know that $H\{G/Y\}\{\text{fix } XF/X\} \xrightarrow{a} Q'$, with $P' \mathcal{R} \sim Q'$; by manipulating substitutions and applying the recursion rule we obtain $G\{\text{fix } XF/X\} \xrightarrow{a} Q'$ as required. □

We now turn to the question of unique fixed-points, up to strong congruence. First, fixed-points always exist:

Proposition 4.7. fix $\tilde{X}\tilde{E}$ *satisfies the formal equations* $\tilde{X} \sim \tilde{E}$ *in* $\tilde{X}$; *that is*, fix $\tilde{X}\tilde{E} \sim \tilde{E}\{\text{fix } \tilde{X}\tilde{E}/\tilde{X}\}$.

Proof. Immediate since, for each i, $\text{fix}_i \tilde{X}\tilde{E} \xrightarrow{a} E'$ iff $E_i\{\text{fix } \tilde{X}\tilde{E}/\tilde{X}\} \xrightarrow{a} E'$, from the recursion rule. □

For what expressions $\tilde{E}$ is this solution unique? A simple sufficient condition is that $\tilde{E}$ is *guarded* in the variables $\tilde{X}$, as we shall show.

Definition. X is *guarded in* E if every free occurrence of X in E is within a subexpression $a:F$ of E.

Examples. X is guarded in $a:X$, fix XX and $Y \times \mathbf{1}$ but not in X, $a:X + X$ or fix $Y(a:Y \times X)$.

Intuitively, an expression which replaces a variable guarded in E cannot contribute to the first action of E; this is expressed by the following simple lemma.

Lemma 4.8. *If* $E\{\tilde{F}/\tilde{X}\} \xrightarrow{a} E'$, *and the variables* $\tilde{X}$ *are guarded in* E, *then for some* E'', $E' \equiv E''\{\tilde{F}/\tilde{X}\}$ *and for any* $\tilde{G}$

$$E\{\tilde{G}/\tilde{X}\} \xrightarrow{a} E''\{\tilde{G}/\tilde{X}\}.$$

Proof. Straightforward by induction on inference. □

Proposition 4.9. *Let* $\tilde{X}$ *be guarded in* $\tilde{E}$, *and let* $\tilde{F} \sim \tilde{E}\{\tilde{F}/\tilde{X}\}$ *and* $\tilde{G} \sim \tilde{E}\{\tilde{G}/\tilde{X}\}$; *then* $\tilde{F} \sim \tilde{G}$.

Proof. It is clearly enough to consider the case in which E has no free variables other than $\tilde{X}$, and in which $\tilde{F}$ and $\tilde{G}$ are agents. Then we can show that

$$\mathcal{R} = \{(H\{\tilde{F}/\tilde{X}\}, H\{\tilde{G}/\tilde{X}\}) \mid \mathrm{Free}(H) \subseteq \tilde{X}\}$$

is a bisimulation up to '$\sim$', from which the result follows by setting $H \equiv X_i$ for each $i \in I$.

For this, we show by induction an inference that if $H\{\tilde{F}/\tilde{X}\} \xrightarrow{a} P'$, then $H\{\tilde{G}/\tilde{X}\} \xrightarrow{a} Q'$ for some Q', with $P' \sim \mathcal{R} \sim Q'$. The inductive step requires analysis of the form of H, and the argument is routine except when $H \equiv X_i$. In this case by assumption $F_i \xrightarrow{a} P'$, hence $E_i\{\tilde{F}/\tilde{X}\} \xrightarrow{a} P'' \approx P'$, and the rest follows simply by Lemma 4.8. □

Remark. In a set of formal equations $\tilde{X} \sim \tilde{E}$ the expression $\tilde{E}$ may not be guarded. However, if they may be converted to guarded equations by a finite amount of 'unwinding'—that is, by repeatedly applying the substitution $\{\tilde{E}/\tilde{X}\}$ to the right-hand sides)—then Proposition 4.9 will again hold for $\tilde{E}$, as is easily verified.

5. Equational properties

Having established our strong congruence '$\sim$', we may now consider the quotient $\mathcal{P}/\sim$ as an algebra and determine a useful set of identities. These, together with the unique fixed-point result (Proposition 4.8), provide considerable power of proof; very often they suffice in showing that two agents are congruent, though one may also revert to the basic technique of finding a bisimulation.

We cannot hope for a complete set of equational laws, nor indeed *any* complete axiomatisation of $\mathcal{P}/\sim$; this is true even for finitely presented agents employing only finite summation and single recursion. For it is rather easy to encode any Turing machine T, starting on blank tape, as an agent P_T; a simple method is to encode the tape as a pair of stacks (along the lines of [8, Section 8.4]) with a separate component agent for T's control. This can be done so that $P_T \sim \mathbf{1}$ iff T diverges on blank input tape, from which we easily deduce that $\{P \mid P \sim \mathbf{1}\}$ is not

recursively enumerable. Restricted or derived calculi may of course be completely axiomatizable, and even admit a decision procedure; this is done by Hennessy [3] for a version without recursion, and by Milner [10] for a version without product or restriction (essentially finite-state nondeterministic automata, but under a finer congruence than that of the classical interpretation in regular sets).

All the following laws are easily established by bisimulations, and we omit proofs after giving one example.

The first set of laws establishes that $\mathscr{P}/\sim$ is a commutative semiring under product and sum, with also an absorption law for sum.

Proposition 5.1. (1) $(\mathscr{P}/\sim, \times, \mathbf{1})$ *is an Abelian monoid.*

(2) $(\mathscr{P}/\sim, +, \mathbf{0})$ *is an Abelian monoid.*

(3) $P\times\mathbf{0} \sim \mathbf{0}$, *and* $P\times(Q+R) \sim P\times Q+P\times R$.

(4) $P+P\sim P$.

Proof. For the second part of (3), for example, establish the bisimulation

$$\{(P\times(Q+R), P\times Q+P\times R)\}\cup \mathrm{Id}_{\mathscr{P}}. \qquad \square$$

The above laws concern only finite summation. In fact, 5.1(2)–(4) are special cases of the following general summation laws.

Proposition 5.2

(1) $\sum_{i\in 1} P \sim P$.

(2) $\sum_{i\in I}\sum_{j\in J_i} P_{ij} \sim \sum_{(i,j)\in K} P_{ij}$.

(3) $\sum_{i\in I} P_i \sim \sum_{j\in J} Q_j$ *if* $\{P_i \mid i\in I\} = \{Q_j \mid j\in J\}$.

(4) $P\times\sum_i Q_i \sim \sum_i (P\times Q_i)$.

There is no simple law relating action 'a:' to summation; it is easy to see that $a:(P+Q) \not\sim a:P+a:Q$ (take $P \equiv b:\mathbf{0}$ and $Q\equiv\mathbf{0}$ for example); this is what distinguishes our interpretation from that of subsets of Act*, which would correspond to classical automata theory. To relate action with product, we have

Proposition 5.3. $a:P\times b:Q \sim ab:(P\times Q)$.

To relate the fourth basic operator, restriction, to the other three we have the following:

Proposition 5.4

(1) $(a:P)\restriction A \sim \begin{cases} a:(P\restriction A) & \text{if } a \in A, \\ \mathbf{0} & \text{otherwise.} \end{cases}$

(2) $\left(\sum_i P_i\right)\restriction A \sim \sum_i (P_i \restriction A).$

(3) $P\restriction A\restriction B \sim P\restriction (A \cap B).$

It is important to note that, in general, $(P \times Q)\restriction A \not\sim P\restriction A \times Q\restriction A$. Indeed, restriction gives a kind of *scoping*; it delimits the scope of certain possibilities of interaction. To deny this distributive law is just to admit that scoping is semantically significant. But the law *does* hold in the special case where, informally speaking, P and Q cannot interact through A; we shall make this precise later.

W now turn to properties of the derived operations, delay and morphism. For delay δ, it is important to see why

$$\delta(P+Q) \not\sim \delta P + \delta Q,$$

$$\delta(P \times Q) \not\sim \delta P \times \delta Q,$$

that is, delay does not distribute over sum and product. For the first, take $P \equiv a:\mathbf{0}$, $Q \equiv b:\mathbf{0}$, so that $\delta(P+Q)$ persistently offers a choice between actions a and b; on the other hand we have $\delta P + \delta Q \xrightarrow{1} \delta P$, an 'autonomous' rejection of δQ, and indeed $\delta(P+Q)$ has no successor congruent to δP. For the second, note that in $\delta(P \times Q)$ the first actions of P and Q will be synchronized whenever they occur, while in $\delta P \times \delta Q$ the delays are independent. However, Proposition 5.5(1) is a weak distributive law, important for later sections.

Proposition 5.5

(1) $\delta P \times \delta Q \sim \delta(P \times \delta Q \stackrel{\cdot}{+} \delta P \times Q).$

(2) $(\delta P)\restriction A \sim \delta(P\restriction A).$

(3) $\delta\delta P \sim \delta P.$

(4) $\delta P \sim P + 1:\delta P \sim P + \delta P.$

For morphism, the situation is rather simple; it distributes over all operators, in some sense.

Proposition 5.6

(1) $(a:P)[\phi] \sim \phi(a):P[\phi].$

(2) $\left(\sum_i P_i\right)[\phi] \sim \sum_i (P_i[\phi]).$

(3) $(P \times Q)[\phi] \sim P[\phi] \times Q[\phi].$

(4) $(P\restriction \phi^{-1}(A))[\phi] \sim (P[\phi])\restriction A.$[1]

[1] This law was incorrectly stated in [9, Proposition 7.5]; Proposition 14.6 of that paper was also wrong. I am grateful to Kevin Mitchell for pointing this out.

(5) $(\delta P)[\phi] \sim \delta(P[\phi])$.

(6) $P[\phi][\psi] \sim P[\psi \circ \phi]$.

(7) $P[\mathrm{Id}_{\mathrm{Act}}] \sim P$.

For the remainder of this section we are concerned with the case in which Act is freely generated, as illustrated in Section 3. We show that agents may be sorted, according to the particulate actions which they employ, and that further useful equations are thus obtained. The later sections on asynchrony do not depend upon these results, though Section 9 uses a freely generated Act.

Particulate actions and sorts: For applications, it is often useful to assume that Act is freely generated by a set Σ of *names*. Let us define, for any $A \subseteq \mathrm{Act}$, $\bar{A} = \{\bar{a} \mid a \in A\}$; then $\bar{\Sigma} = \{\bar{\sigma} \mid \sigma \in \Sigma\}$ are the *conames*, and we call $\Lambda = \Sigma \cup \bar{\Sigma}$ the *particles, or particulate actions*. This brings us closer to CCS [8], where we called Λ the labels; there, however, the only actions were $\Lambda \cup \{1\}$.

Let us say that particles λ, λ' are *independent* if $\lambda \neq \lambda' \neq \bar{\lambda}$. Then every action a is uniquely expressible (up to order of factors) as a product $\lambda_1^{n_1}\lambda_2^{n_2} \cdots \lambda_k^{n_k}$ $(n_i > 0)$ of powers of independent particles; we denote $\{\lambda_1, \ldots, \lambda_k\}$ by $\mathrm{Part}(a)$. Also for $A \subseteq \mathrm{Act}$ we set $\mathrm{Part}(A) = \bigcup \{\mathrm{Part}(a) \mid a \in A\}$.

Now it can be useful to classify agents according to the actions which they may perform. For this purpose we choose the submonoids generated by subsets of Λ.

Definition. For $L \subseteq \Lambda$, denote by L^* the submonoid of Act generated by L.

Definition. For each $L \subseteq \Lambda$, let $\mathcal{P}_L$ be the set of agents P such that for any derivative Q of P, if $Q \xrightarrow{a} Q'$ then $a \in L^*$. If $P \in \mathcal{P}_L$ we say P has *sort* L, *or write* $P::L$. Thus if $P::L$, then $\mathrm{Part}(a) \subseteq L$ for any action a of a derivative of P.

Other classifications are possible. An obvious candidate is based upon subgroups, not submonoids, of Act; this gives a coarser classification.

A convenient form of restriction $\upharpoonright A$ is the case where A is a submonoid; as we did in Section 3, we write $\backslash N$ for $\upharpoonright (\Lambda - N)^*$, and assume for the remainder of this section that only such restrictions are employed.

The basic and derived operations act upon sorts as follows:

Proposition 5.7. (1) *If* $P::L$ *and* $L \subseteq M$ *then* $P::M$.

(2) *If* $a \in L^*$ *and* $P::L$ *then* $a:P::L$.

(3) *If* $\tilde{P}::L$ *then* $\sum \tilde{P}::L$.

(4) *If* $P, Q::L$ *then* $P \times Q::L$.

(5) *If* $P::L$ *then* $P\backslash N::(L - N)$.

(6) *If* $P::L$ *then* $\delta P::L$.

(7) *If* $P::L$ *then* $P[\phi]::Part(\phi(L))$.

To determine a sort for an arbitrary agent, we may use the following.

Definition. A *sorting* of an agent P is an ascription of a sort, $\mathrm{Sort}(X)$, to each variable at all its occurrences, and a sort $\mathrm{Sort}(E)$ to each subexpression E of P,

such that

(1) $\mathrm{Sort}(a:F) = \mathrm{Part}(a) \cup \mathrm{Sort}(F)$,
(2) $\mathrm{Sort}(\sum_i F_i) = \bigcup_i \mathrm{Sort}(F_i)$.
(3) $\mathrm{Sort}(F \times G) = \mathrm{Sort}(F) \cup \mathrm{Sort}(G)$.
(4) $\mathrm{Sort}(F\backslash N) = \mathrm{Sort}(F) - N$.
(5) $\mathrm{Sort}(\mathrm{fix}_i \tilde{X}\tilde{F}) = \mathrm{Sort}(X_i)$, and for each j $\mathrm{Sort}(F_j) \subseteq \mathrm{Sort}(X_i)$.
(6) $\mathrm{Sort}(\delta F) = \mathrm{Sort}(F)$.
(7) $\mathrm{Sort}(F[\phi]) = \mathrm{Part}(\phi(\mathrm{Sort}(F))$.

Note that the sort ascribed to each variable must be large enough to meet the side condition in (5). It is easy to show that a sorting may always be found as the limit of an iteration, but we leave aside the question of conditions under which a finite sort can be reached by a finite iteration. In practice, sortings are easy to discover. We now claim that the definition is sound, but omit the proof, which is straightforward.

Proposition 5.8. *If L is ascribed to an agent P in any sorting of P, then $P::L$.*

We now give some equational laws which hold only under limitations on sort.

Definition. A morphism ϕ is *faithful on L* if $1 \notin \phi(L)$ and, whenever $\lambda, \lambda' \in L$ are independent, $\mathrm{Part}(\phi(\lambda))$ and $\mathrm{Part}(\phi(\lambda'))$ are pairwise independent.

For example, if $\phi(\alpha) = \alpha'\lambda$ and $\phi(\beta) = \beta'\bar{\lambda}$ then ϕ is not faithful on $\{\alpha, \beta\}$; but the morphisms ϕ, ψ of Section 3 are faithful on appropriate sorts.

Proposition 5.9. (1) $P\backslash N \sim P$, *if* $P::L$ *and* $L \cap N = 0$.
(2) $P[\phi] \sim P[\psi]$, *if* $P::L$ *and* $\phi \restriction L = \psi \restriction L$.
(3) $(P \times Q)\backslash N \sim P\backslash N \times Q\backslash N$, *if* $P::L$, $Q::M$ *and* $L \cap \bar{M} \cap N = \bar{L} \cap M \cap N = 0$.
(4) $P\backslash N[\phi] \sim P[\phi]\backslash \mathrm{Part}(\phi N)$, *if* $P::L$ *and* ϕ *is faithful on* $L \cup N$.

The third law allows a restriction $\backslash N$ to distribute over product only when the factors are unable to 'communicate' via a member of N.

A corollary of the fourth law shows how restriction may be used to delimit the scope of a particle. Let us define

$$\backslash\!\!\backslash N = \backslash(N \cup \bar{N})$$

so that both a particle and its inverse are restricted. This is the form of restriction used in CCS. Also let λ'/λ stand for the morphism which is the identity on L except that $\lambda \mapsto \lambda'$, $\bar{\lambda} \mapsto \bar{\lambda}'$.

Corollary 5.10 (Restricted relabelling). *If* $P::L$ *and* $\lambda', \bar{\lambda}' \notin L$, *then*

$$P\backslash\!\!\backslash \lambda \sim P[\lambda'/\lambda]\backslash\!\!\backslash \lambda'.$$

There is a close analogy with the ubiquitous transformation known as 'change of bound variables'; such transformations are the essence of syntactic scope. Note that the scope of particles, delimited by restriction, is entirely independent of the scope of agent variables X, delimited by the recursion operator fix.

6. Further constructions and discussion

In this section we wish to show how in SCCS indexed summation, indexed recursion and non-particulate actions allow constructions both in and beyond CCS. Of course synchrony is beyond CCS, but what follows applies to both synchrony an asynchrony.

CCS possesses the constructs $\alpha x.E$ and $\bar{\alpha}v.F$, for input and output of values in an arbitrary data domain D. Here x must be a variable of type D, and its occurrence in '$\alpha x.$' is a binding occurrence, while v may be any value expression of type D. Now, by employing D-indexed families of particles α_d and $\bar{\alpha}_d$, $d \in D$, we may take $\alpha x{:}E$ and $\bar{\alpha}v{:}F$ to abbreviate $\sum_{x \in D} \alpha_x{:}E$ and $\bar{\alpha}_v{:}F$ in SCCS. (Also, as we see in Part II, $\lambda.E$ in CCS corresponds to $\lambda{:}\delta(E)$ in SCCS.) This was illustrated in the Flip-flop example, Section 3; what is important is that we do not require sets indexing summation to be finite, so any data domain may be so treated. Moreover, if (as in CCS) E takes the form of a conditional expression '**if** b **then** E_0 **else** E_1' where b is a boolean valued expression in x, then we have the encoding

$$\alpha x{:}E \rightarrow \sum_{x \in D \& b} \alpha_x{:}E_0 + \sum_{x \in D \& \neg b} \alpha_x{:}E_1.$$

Such methods can be extended, using the indexed recursion $\mathrm{fix}_i \tilde{X}\tilde{E}$ of SCCS, to encompass also the recursive definition of parametric agents, as illustrated at the end of the Flip-flop example. To give a formal encoding of all CCS would be excessive here; we merely point out that even though the expressions of CCS are more convenient than their encodings, for semantic purposes it is sufficient to study the more primitive calculus of this paper.

We now consider something which cannot obviously be expressed in a general form in CCS: the passing of communication links as values, allowing dynamic reconfiguration of agent linkage (cf. [8, end of Section 9.5]). We will treat a simple case, corresponding to a program procedure which can support unboundedly many concurrent activations. The agent

$$Q \equiv \delta(\beta x{:}\delta(\bar{\gamma}f(x){:}\mathbf{1}))$$

(using the previous construction) will compute the value $f(v)$ for arbitrary input v. One might define recursively

$$P \Leftarrow \delta(\alpha{:}(Q \times P))$$

which, when excited at α, creates a copy of Q to run concurrently with itself. A

third agent R may contain calling sequences

$$\ldots \bar{\alpha}:\bar{\beta}v:\ldots \gamma y:\ldots$$

asking for a creation of Q, then calling Q with value v and later receiving the value $f(v)$ bound to y. Now the composite system

$$(P \times R)\backslash\backslash\{\alpha, \beta, \gamma\}$$

will work well if R's calling sequences are never executed concurrently, since P will never create a second copy of Q until the first has been 'executed'. But it will fail for concurrent calling sequences, since values $f(v)$ may be sent back to the wrong callers.

In CCS it is easy enough to handle correctly a bounded number of concurrent activations of Q, but not obviously an unbounded number. In SCCS, we first introduce ω-indexed families β_i, γ_i of particles, and define morphisms $\phi_i = \beta_i/\beta$, γ_i/γ for $i \in \omega$. Next, we redefine P as P_0, where for $i \in \omega$

$$P_i \Leftarrow \delta(\alpha:(Q[\phi_i] \times P_{i+1}))$$

so that a distinctly labelled copy of Q is created for each call. Finally, we amend R's calling sequences to

$$\bar{\alpha}: \sum_{i \in \omega} (\bar{\beta}_i v: \ldots \gamma_i y: \ldots).$$

The effect is that as soon as v is passed to some $Q[\phi_i]$, the calling sequence 'knows' i and receives the correct return value $f(v)$ at γ_i. The critical point is not so much the use of infinite sum as the dependence (via ϕ_i) of the actions of P_i upon the index i. Notice that P_i now has an infinite sort $\{\beta_i, \bar{\gamma}_i \mid j \geq i\}$; we have not been forced to abandon sorting, which will be essential in reasoning about systems. But it is possible to represent systems in which the sort of an agent depends upon its activity in much more sophisticated ways, and we do not claim that it will be easy to analyse them.

Turning to the question of modelling communication, we first note that in SCCS the construct $(P \times Q)\backslash\backslash\alpha$ causes each α action of P to be synchronized with an $\bar{\alpha}$ action of Q; this synchronization of communication between *two* agents is basic also in CCS. In CCS however, it is not possible to achieve *multi-way* synchronization (of an arbitrary finite number of agents) in an elementary way, while in SCCS it is easily done using non-particulate actions, by such constructs as

$$(\ldots \alpha: \ldots) \times (\ldots \beta: \ldots) \times (\ldots \bar{\alpha}\bar{\beta}: \ldots))\backslash\backslash\{\alpha, \beta\}.$$

Following ideas of Hoare [4] and Milne [12], we further consider a binary operator $\&_\gamma$, which we may call *γ-conjunction*, and which behaves like product except that in

$$P_1 \&_\gamma \cdots \&_\gamma P_n$$

each γ-action of P_i must synchronize with a γ-action of *every* P_i. The operator could be introduced by action rules:

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q'}{P \&_\gamma Q \xrightarrow{ab} P' \&_\gamma Q'}, \qquad \frac{P \xrightarrow{a\gamma} P' \quad Q \xrightarrow{b\gamma} Q'}{P \&_\gamma Q \xrightarrow{ab\gamma} P' \&_\gamma Q'}$$

for $a, b \in (\Lambda - \{\gamma, \bar{\gamma}\})^*$. But the operator can be derived. We first define a γ-synchronizer, using two *new* names α and β:

$$\mathrm{SYN}_\gamma \Leftarrow \delta(\bar{\alpha}\bar{\beta}\gamma{:}\mathrm{SYN}_\gamma).$$

Then the required operator is defined

$$P \&_\gamma Q \equiv (P[\alpha/\gamma] \times Q[\beta/\gamma] \times \mathrm{SYN}_\gamma) \backslash\backslash \{\alpha, \beta\}$$

which is justified by the following:

Proposition 6.1. *For $n \geq 2$, $P_1 \&_\gamma \cdots \&_\gamma P_n \xrightarrow{a} P'$ iff*

(i) *$P' \equiv P_1' \&_\gamma \cdots \&_\gamma P_n'$ and*

(ii) *for some $a_i \in (\Lambda - \{\gamma, \bar{\gamma}\})^*$, $1 \leq i \leq n$,*

either $a = a_1 \cdots a_n$ and $P_i \xrightarrow{a_i} P_i'$, $\quad 1 \leq i \leq n$

or $\quad a = a_1 \cdots a_n \gamma$ and $P_i \xrightarrow{a_i \gamma} P_i'$, $\quad 1 \leq i \leq n$.

Further, γ-conjunction generalises easily to Γ-conjunction for any subset $\Gamma \subseteq \Sigma$ of particles; $\&_\Gamma$ requires synchronization on any $\gamma \in \Gamma$. The operators $\&_\Gamma$ are well-behaved, as is shown by the following proposition, which is a good exercise in using the algebraic laws (the reader may like to check it in the special case $\Gamma = \{\gamma\}$).

Proposition 6.2. (1) *$(\mathcal{P}, \&_\Gamma)$ is an Abelian Semigroup.*

(2) *$P \&_\Gamma (Q + R) \sim P \&_\Gamma Q + P \&_\Gamma R$.*

We do not have a monoid, since there is no unit for $\&_\Gamma$. I am grateful to both T. Denvir and to the referee for pointing out that **1** is not a unit. However, in a natural subcalculus (in which, roughly speaking, Γ is only used for multiway synchronization) it turns out that

$$\mathbf{1}_\Gamma \equiv \mathrm{fix}\, X\Big(1{:}X + \sum_{\gamma \in \Gamma} \gamma{:}X\Big)$$

is a unit, and that $P \&_\Gamma P \sim P$ also holds—in contrast with simple product.

There is good reason not to take the operators $\&_\Gamma$ as primitive, in place of product. The reason is that there is one operator for each set $\Gamma \subseteq \Sigma$, and the conjunction $P \& Q$ is *not* well-defined unless Γ is supplied. By convention, Γ may perhaps be taken to be $L \cap M$, where L and M are the respective minimum sorts

of P and Q (this means that P and Q must synchronize upon any action which is 'possible' for both). But then the undecorated conjunction '&' is no longer associative! Moreover, in a derivation $P \& Q \xrightarrow{a} P' \& Q'$ the minimal sorts of P' and Q' may differ from those of P and Q, and there is danger of confusion. Hence the whole family $\&_I$ of conjunctions must be considered, and indeed the laws which relate them all are quite complex. But if a system can be expressed conveniently using $\&_I$ for a few particular sets I, and without using product, then it may well be an advantage to do so.

Before proceeding to Part II, in which SCCS is shown to underly asynchronous communication, let us try to summarise its essential features. First, it emphasizes the *interfaces* between agents; since the character of a composite system depends at least as much upon the organization of its parts as upon the behaviour of the parts themselves, the author feels that one cannot escape from attempting to mathematize this organization, which is largely to do with interfaces, and that the Action group is a significant step in this direction.

Second, time is modelled discretely. It is not clear how limiting this is: by choosing a small enough time-grain one could presumably treat any synchronous system, using convenient abbreviations to represent explicit delays $(1:)^n$. A bounded delay could be written δ_n, where $\delta_0(P) \equiv P$ and $\delta_{n+1}(P) \equiv P + 1:\delta_n(P)$. Recently Cardelli [1] has adapted the calculus to a time domain which is a dense order ('real' time rather than 'integer' time), and a delicate comparison of the calculi needs to be made. On the other hand, the interfaces in SCCS (the Action space) need not be discrete; for example, the idealised Flip-flop of Section 3 can be made less 'ideal' is we model voltage by families λ_r of particles, where r ranges over the unit interval $[0, 1]$.

Third, the calculus is much more general than our two small examples may suggest (though the remark in Section 5 that it cannot be recursively axiomatized does suggest so). Unbounded replication of agents is possible, by a mixture of product and recursion; an example was given earlier in this section. An apparent limitation is the lack of higher-order objects; there are no 'agents which process agents'. But it is far from clear what these objects could be, or whether they are needed. Consider FINDROOT(f, a, b), a functional for finding a zero of the function f in the range (a, b); it is needed in programming, and can be programmed in an applied typed lambda-calculus using second order functions. But in SCCS, or in its asynchronous derived calculus CCS, FINDROOT is just an agent, part of whose interface is designed to be connected to an agent for computing the function f. Indeed, this example was given in [8, Chapter 4]. Apparently—at least in this example—the concept of interactive *process* (in contrast to mathematical *function*) can render higher-order notions unnecessary.

Finally, the semantic treatment of SCCS is founded on operational rules. The reason for this was discussed in Section 1, and we merely conclude by saying that this treatment opens the way to analysing other interpretations, which undoubtedly exist, in relation to the more basic notion of action.

PART II. ASYNCHRONY

7. An asynchronous subcalculus

In this section we introduce a definition of asynchrony as a property of agents, and show that, in a certain subcalculus of SCCS, only asynchronous agents are definable. Here and in Section 8 we assume that Act is any Abelian group.

In what sense are the agents $\mathscr{P}$ synchronous? Consider for example the agent

$$a:b:\mathbf{0}.$$

It must perform the action b immediately after a, with no time delay. Similarly, if we regard $1 \in \mathrm{Act}$ as a silent or waiting action, then $a:1:b:\mathbf{0}$ performs b with a delay of exactly one instant after performing a. One can think of the agents as synchronized with a universal clock.

By contrast, the agent $a:\delta(b:\delta\mathbf{0})$ may idle indefinitely between a and b – and idle for ever thereafter. We may call δP an idle agent. But we want idleness, and other properties, to be preserved by strong congruence, so we make the following

Definition. P is *idle* if $P \sim \delta P$.

The following is easily established, with the help of Proposition 5.5(3).

Proposition 7.1. (1) *δP is idle, for any P.*
(2) *If $P \xrightarrow{1} P$ then P is idle.*
(3) *If $P \sim Q$ and P is idle, then Q is idle.*

In view of the above discussion, it would now be natural to take an asynchronous agent P to be one whose derivatives are all idle. But we wish to make one reservation; we shall not require that *P itself* be idle. The reason is that we wish, by summation, to be able to build even asynchronous agents whose course of action can be externally controlled. For example $a:P$ and $b:Q$ are not idle, and their sum $a:P+b:Q$ is very different from the sum $\delta(a:P)+\delta(b:Q)$ of idle agents, since the latter possesses the invisible action

$$\delta(a:P)+\delta(b:Q) \xrightarrow{1} \delta(a:P)$$

by which the second summand is autonomously discarded. By contrast, one or other summand of $a:P+b:Q$ may only be discarded as a result of a visible action; it is externally controllable. We therefore define

Definition. P is *asynchronous* if all its proper derivatives are idle.

It is easy to see that this property also is preserved by strong congruence. To what extent is it preserved by the operators of SCCS? The action operator '$a:$' does not preserve it; $b:\delta\mathbf{0}$ is asynchronous (its only proper derivative is $\delta\mathbf{0}$) but $a:b:\delta\mathbf{0}$ is not. To get an asynchronous calculus, we *remove* this operator, but *replace* it by the derived asynchronous action operator '$a.$', defined by

$$a.E \equiv a:\delta E.$$

It is technically convenient to include the delay operator δ in our derived calculus, even though it can be defined (up to $\sim$) in terms of the asynchronous action operator, since it is easily shown that $\delta E \sim \mathrm{fix}\, X(1.X + E)$. The morphism operators can also be defined equally well using '$a.$' in place of '$a:$', and we shall not include them. So our asynchronous derived calculus, which we call ASCCS, has the syntax

$$E ::= X \mid a.E \mid \delta E \mid \sum \tilde{E} \mid E \times E \mid E \restriction A \mid \mathrm{fix}_i \tilde{X}\tilde{E}.$$

Let $\mathscr{E}^{\mathrm{AS}}$ and $\mathscr{P}^{\mathrm{AS}}$ be respectively the expressions and the agents (closed expressions) of ASCCS. We now show that ASCCS is indeed asynchronous.

Proposition 7.2. (1) *If* $P \in \mathscr{P}^{\mathrm{AS}}$ *and* $P \xrightarrow{a} P'$, *then* $P' \in \mathscr{P}^{\mathrm{AS}}$ *and* $P' \xrightarrow{1} P'$.
(2) *Every* P *in* $\mathscr{P}^{\mathrm{AS}}$ *is asynchronous.*

Proof. The second part follows directly from the first, using Proposition 7.1(2), and we prove (1) by induction on inference. For the inductive step, assume $P \xrightarrow{a} P'$ and consider the cases for P. If $P \equiv b.Q$, then $b = a$ and $P' \equiv \delta Q$; the result is immediate. In all other cases the induction is routine. □

Remark. As suggested by a referee, it would be pleasant to show a converse to Proposition 7.2(2), namely that every asynchronous agent is expressible (up to $\sim$) in ASCCS. This is probably true, but for the present, we leave the question open.

8. Observation congruence in the asynchronous calculus

We shall now show that even if two asynchronous agents P and Q differ in the amount of delay (invisible actions) which intervenes among their visible actions, they cannot be distinguished—in whatever asynchronous context we place them—by an observer who cannot measure the length of a sequence of invisible actions. An example will make this clearer. Let $P \equiv a.b.\mathbf{0}$ and $Q \equiv a.1.b.\mathbf{0}$; then although both P and Q are capable of indefinite delay between the actions a and b, P *can* perform b immediately following a, while Q *cannot*. Now P cannot be directly distinguished from Q by an observer with the above-mentioned incapacity, but even such an observer can distinguish them in a synchronous context; if $R \equiv c:d:\mathbf{0}$ (synchronous!), then $P \times R$ may be observed to perform the action ac followed by bd, and $Q \times R$ may not be so observed.

Now P and Q are not strongly congruent, but it is necessary to provide a form of asynchronous equivalence containing such pairs $\langle P, Q\rangle$ since we are often concerned only with the visible behaviour of a system. We shall define this equivalence, which we call *observation equivalence*, by analogy with strong congruence, and show that it yields—by slight strengthening—an *observation congruence* relation for ASCCS. The spirit is the same as in CCS [8, Chapter 7]) though the details differ. In Section 9 we show how ASCCS may be further constrained to become isomorphic to CCS.

Sequences of visible actions are members of $(\mathrm{Act}-\{1\})^*$. To represent the observation of such a sequence, with arbitrary intervening 1-actions, we make the following definition.

Definition. Let $u = \langle a_1, a_2, \ldots, a_n\rangle \in \mathrm{Act}^*$. Then

$$P \overset{u}{\Rightarrow} P' \quad \text{iff} \quad P(\overset{1}{\rightarrow})^* \overset{a_1}{\rightarrow} (\overset{1}{\rightarrow})^* \cdots \overset{a_n}{\rightarrow} (\overset{1}{\rightarrow})^* P'.$$

Note the special case $u = \wedge$, the empty sequence. We shall often write $P \Rightarrow P'$ for $P \overset{\wedge}{\Rightarrow} P'$, i.e. for $P(\overset{1}{\rightarrow})^* P'$; in particular we have $P \Rightarrow P$. On the other hand $P \overset{1}{\Rightarrow} P'$ stands for $P(\overset{1}{\rightarrow})^* \overset{1}{\rightarrow} P'$. But we shall usually have $u \in (\mathrm{Act}-\{1\})^*$.

Definition. A binary relation $\mathcal{S} \subseteq \mathcal{P}^2$ is a *weak* (or *observational*) *bisimulation* if, whenever $P\mathcal{S}Q$ and $u \in (\mathrm{Act}-\{1\})^*$,

(i) if $P \overset{u}{\Rightarrow} P'$ then, for some Q', $Q \overset{u}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$.

(ii) if $Q \overset{u}{\Rightarrow} Q'$ then, for some P', $P \overset{u}{\Rightarrow} P'$ and $P'\mathcal{S}Q'$.

As in the case of strong bisimulation, we denote by $\mathcal{G}(\mathcal{S})$ the set of pairs $\langle P, Q\rangle$ satisfying clauses (i) and (ii), and then the definition requires that $\mathcal{S} \subseteq \mathcal{G}(\mathcal{S})$. Analogous propositions follow.

Proposition 8.1. *$\mathcal{G}$ is monotonic over the lattice of relations under inclusion.*

Proposition 8.2. *There is a maximum weak bisimulation* $\approx = \bigcup\{\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{G}(\mathcal{S})\}$. *Moreover* $\approx = \mathcal{G}(\approx)$, *and '$\approx$' is the maximum fixed point of $\mathcal{G}$.*

Proposition 8.3. *'$\approx$' is an equivalence relation.*

We shall call '$\approx$' *weak* (or *observation*) equivalence.

The reader may naturally ask why we did not restrict consideration to $\overset{a}{\Rightarrow}$, $a \in \mathrm{Act}-\{1\}$, in our definition. The reason is that $\overset{\wedge}{\Rightarrow}$, or $\Rightarrow$, is also important. However, in our definition we could have restricted u to range over sequences of length ≤ 1 without changing matters. The following easy result allows us to work conveniently.

Proposition 8.4. *$\mathcal{S}$ is a weak bisimulation iff, whenever $P\mathcal{S}Q$ and $a \in$ Act,*

(i) *if $P \xrightarrow{a} P'$ then* either $a = 1$ *and $P'\mathcal{S}Q$* or, *for some Q', $Q \stackrel{a}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$,*

(ii) *if $Q \xrightarrow{a} Q'$ then* either $a = 1$ *and $P\mathcal{S}Q'$*, or, *for some P', $P \stackrel{a}{\Rightarrow} P'$ and $P'\mathcal{S}Q'$.*

Thus the difference from strong bisimulation is apparent; each single (visible or invisible) action of P must be matched not by a single action of Q, but by an action sequence of Q with the same visible content Using this result, we can easily establish

Proposition 8.5. *$P \sim Q$ implies $P \approx Q$.*

Proof. Every strong bisimulation is a weak one, by Proposition 8.4 □

In contrast to strong congruence, we also have

Proposition 8.6. *$P \approx 1{:}P \approx 1.P \approx \delta P$.*

Proof. For example, $\{\langle P, 1{:}P\rangle \mid P \in \mathcal{P}\} \cup \mathrm{Id}_{\mathcal{P}}$ is a weak bisimulation. □

We must now discover to what extent '$\approx$' is a congruence. Certainly some of the operators of SCCS preserve it:

Proposition 8.7. *If $P \approx Q$ then $a{:}P \approx a{:}Q$, $\delta P \approx \delta Q$ and $P \upharpoonright A \approx Q \upharpoonright A$.*

Proof. For example, $\{\langle a{:}P, a{:}Q\rangle \mid P \approx Q\} \cup \approx$ is a weak bisimulation, using Proposition 8.4. □

However, $P \approx Q$ does not imply $P \times R \approx Q \times R$ even within ASCCS: To see this, take $P \equiv a.\mathbf{0}$, $Q \equiv 1.a.\mathbf{0}$, $R \equiv b.\mathbf{0}$. The action ab is observable for $P \times R$ but not $Q \times R$. Nor does $P \approx Q$ imply $P + R \approx Q + R$, even within ASCCS; the same values P,Q,R provide a counter example. We must therefore look for the largest congruence relation contained in $\approx$; on general grounds, this should be the relation $\approx'$ defined by

$$P \approx' Q \quad \text{iff} \quad \text{for all contexts } \mathscr{C}[\ ],\ \mathscr{C}[P] \approx \mathscr{C}[Q].$$

Here, a context $\mathscr{C}[\ ]$ is an expression with zero or more 'holes', to be filled by an expression. But the definition of $\approx'$ depends upon which contexts are allowed. If any SCCS context is allowed, it appears that $\approx'$ will be just strong congruence; for by choosing $\mathscr{C}[\ ] \equiv [\ ] \times R$, where R is synchronous, R can be used to detect any delay discrepancy between P and Q. Thus, the whole import of the present section is that by allowing only *asynchronous contexts*, i.e. those of ASCCS, we obtain a congruence $\approx'$ very close to $\approx$.

For the remainder of this section, we are concerned only with the agents, expressions and contexts of ASCCS. First we extend $\approx$ to $\mathscr{E}^{AS}$.

Definition. If $\tilde{X} = \text{Free}(E, F)$ then $E \approx F$ iff, for all agents $\tilde{P}$, $E\{\tilde{P}/\tilde{X}\} \approx F\{\tilde{P}/\tilde{X}\}$.

Definition. $E \approx^c F$ iff for all (asynchronous) contexts $\mathscr{C}[\ ]$, $\mathscr{C}[E] \approx \mathscr{C}[F]$.

The following theorem gives a characterisation of $\approx^c$.

Theorem 8.8. *The following are equivalent in* ASCCS:

(1) $P \approx^c Q$;

(2) *For all* R *in* $\mathscr{P}^{\text{AS}}$, $P \times R \approx Q \times R$;

(3) *For all* $a \in \text{Act}$,

(i) *if* $P \xrightarrow{a} P'$ *then, for some* Q', $Q \xrightarrow{a} \Rightarrow Q'$ *and* $P' \approx Q'$,

(ii) *if* $Q \xrightarrow{a} Q'$ *then, for some* P', $P \xrightarrow{a} \Rightarrow P'$ *and* $P' \approx Q'$.

Moreover $\approx^c$ *is a congruence, and, if* $\tilde{X} = \text{Free}(E, F)$, *then* $E \approx^c F$ *iff, for all agents* $\tilde{P}$, $E\{\tilde{P}/\tilde{X}\} \approx^c F\{\tilde{P}/\tilde{X}\}$.

Before proving the theorem, note that clause (3)—as compared with Proposition 8.4—imposes a stronger condition than weak bisimulation, only upon the initial actions of P and Q. The following is also a direct corollary, using clause (3).

Corollary 8.9. $E \sim F$ *implies* $E \approx^c F$ *implies* $E \approx F$.

The first main step towards the theorem is to show (2) $\Leftrightarrow$ (3). For this, we need the following.

Lemma 8.10. *If* P, Q *and* R *are idle and* $P \approx Q$ *then* $P \times R \approx Q \times R$.

Proof. We show that $\mathscr{S} = \{\langle P \times R, Q \times R\rangle \mid P, Q \text{ and } R \text{ idle}, P \approx Q\}$ is a weak bisimulation for ASCCS, using Proposition 8.4. For a typical pair in $\mathscr{S}$, let $P \times R \xrightarrow{c} P' \times R'$. Then $P \xrightarrow{a} P'$, $R \xrightarrow{b} R'$ and $ab = c$. By Proposition 8.4, *either* $a = 1$ and $P' \approx Q$, and (since Q is idle) $Q \xrightarrow{1} Q$ and $Q \times R \xrightarrow{c} Q \times R'$ with $(P' \times R')\mathscr{S}(Q \times R')$, *or* $Q(\xrightarrow{1})^m \xrightarrow{a} (\xrightarrow{1})^n Q'$ and $P' \approx Q'$. In the latter case (since R and R' are idle) $R(\xrightarrow{1})^m R \xrightarrow{b} R'(\xrightarrow{1})^n R'$, so $Q \times R \Rightarrow Q' \times R'$ with $(P' \times R')\mathscr{S}(Q' \times R')$. □

Lemma 8.11. *In the statement of Theorem* 8, (2) $\Leftrightarrow$ (3).

Proof. ($\Leftarrow$) Assume (3), and let $P \times R \xrightarrow{c} P' \times R'$. Then $P \xrightarrow{a} P'$, $R \xrightarrow{b} R'$ and $ab = c$. By (3), $Q \xrightarrow{a} (\xrightarrow{1})^n Q'$, with $P' \approx Q'$. Since R' is idle, $R'(\xrightarrow{1})^n R'$, so $Q \times R \Rightarrow Q' \times R'$, and $P' \times R' \approx Q' \times R'$ by the previous lemma. With a symmetric argument, (2) is established.

($\Rightarrow$) Assume that (3) is false, so that for example $P \xrightarrow{a} P'$, but whenever $Q \xrightarrow{a} \Rightarrow Q'$ then $P' \not\approx Q'$. Choose $R \equiv (\bar{a}b).\mathbf{0}$ where $a\bar{b}$ is not an action of Q; then $P \times R \xrightarrow{b} P' \times \mathbf{1}$. Now suppose $Q \times R \Rightarrow Q' \times R'$; then we must have $Q \xrightarrow{a} \Rightarrow Q'$ and $R' \equiv \mathbf{1}$, from which it follows that $Q' \times R' \not\approx P' \times \mathbf{1}$. Hence we have found R for which $P \times R \not\approx Q \times R$, and (2) is false. □

Definition. $E \approx^{\times} F$ iff, for all R in $\mathcal{P}^{AS}$, $E \times R \approx F \times R$.

The second main step towards Theorem 8.8 is to show that $\approx^{\times}$ is a congruence; it will then follow easily that it is identical with $\approx^{c}$.

Lemma 8.12. *If* $\tilde{X} = \mathrm{Free}(E, F)$, *then* $E \approx^{\times} F$ *iff, for all agents* $\tilde{P}$, $E\{\tilde{P}/\tilde{X}\} \approx^{\times} F\{\tilde{P}/\tilde{X}\}$.

Proof. Immediate from the definitions. □

Lemma 8.13. $E \approx' F$ *implies* $E \approx F$.

Proof. Take $R \equiv \mathbf{1}$ in the preceding definition. □

Lemma 8.14. *The relation* $\approx'$ *is a congruence in* ASCCS; *that is,*

(1) $E \approx' F$ *implies* $a.E \approx' a.F, \delta E \approx' \delta F, E \times G \approx' F \times G$ *and* $E \restriction A \approx' F \restriction A$;

(2) $\tilde{E} \approx' \tilde{F}$ *implies* $\sum \tilde{E} \approx' \sum \tilde{F}$ *and* $\mathrm{fix}_i \tilde{X}\tilde{E} \approx' \mathrm{fix}_i \tilde{X}\tilde{F}$.

Proof. (1) By the definitions, it is enough to do the proof for agents only. In each case, by Lemma 8.11, we may work with the characterization of clause (3) of the theorem, and the details are routine with the help of Lemma 8.13. To prove $P \approx' Q$ implies $P \times R \approx' Q \times R$ it is even simpler to reason as follows: for every $S \in \mathcal{P}^{AS}$, $(P \times R) \times S \sim P \times (R \times S) \approx Q \times (R \times S) \sim (Q \times R) \times S$.

(2) For summation, we work directly with clause (3) of the theorem, as above. For recursion it will be enough to prove just the simple case

$$E \approx' F \quad \text{implies} \quad \mathrm{fix}\, XE \approx' \mathrm{fix}\, XF$$

when at most X is free in E or F. Let

$$\mathcal{S} = \{\langle G\{\mathrm{fix}\, XE/X\}, G\{\mathrm{fix}\, XF/X\}\rangle \mid \mathrm{Free}(G) \subseteq \{X\}\}.$$

We now wish to prove, by induction on inference, a property which will not only ensure that $\mathcal{S}$ is a weak bisimulation up to $\approx$, from which it would follow (by setting $G \equiv X$) that $\mathrm{fix}\, XE \approx \mathrm{fix}\, XF$, but also that an arbitrary pair in $\mathcal{S}$ has a property like clause (3) of the theorem, allowing us to deduce $\mathrm{fix}\, XE \approx' \mathrm{fix}\, XF$ as required.

In fact we shall show, by induction on inference, that

$$\text{If } G\{\mathrm{fix}\, XE/X\} \xrightarrow{a} P' \text{ then, for some idle } Q' \text{ and } R',$$

$$G\{\mathrm{fix}\, XF/X\} \xrightarrow{a} \Rightarrow R' \text{ and } P' \mathcal{S} Q' \approx R'. \qquad (\star\star)$$

It is clear that this implies $\mathcal{S} \subseteq \mathcal{G}(\approx \mathcal{S} \approx)$ and that (by taking $G \equiv X$) $\mathrm{fix}\, XE \approx' \mathrm{fix}\, XF$ from Lemma 8.11.

Now assume $G\{\text{fix}\, XE/X\} \xrightarrow{a} P'$, and consider cases for G, omitting cases for which the proof of (★★) is as simple.

(i) $G \equiv X$. Then $\text{fix}\, XE \xrightarrow{a} P'$ so by a shorter inference $E\{\text{fix}\, XE/X\} \xrightarrow{a} P'$, and by induction, for some idle Q' and R'', $E\{\text{fix}\, XF/X\} \xrightarrow{a} \Rightarrow R''$ and $P'\mathcal{S}Q' \approx R''$. Since $E \approx^{\times} F$, we deduce from Lemma 8.11 that for some $R' \approx R''$, $F\{\text{fix}\, XF/X\} \xrightarrow{a} \Rightarrow R'$. Hence by the recursion rule

$$G\{\text{fix}\, XF/X\} \equiv \text{fix}\, XF \xrightarrow{a} \Rightarrow R' \quad \text{and} \quad P'\mathcal{S}Q' \approx R'.$$

(ii) $G \equiv G_0 \times G_1$. Then by shorter inferences, $G_i\{\text{fix}\, XE/X\} \rightarrow^{a_i} P_i'$ $(i = 0, 1)$, with $a_0 a_1 = a$ and $P_0' \times P_1' \equiv P'$. By induction, there are idle Q_i' and R_i' such that for $i = 0, 1$,

$$G_i\{\text{fix}\, XF/X\} \xrightarrow{a_i} \Rightarrow R_i' \quad \text{and} \quad P_i'\mathcal{S}Q_i' \approx R_i'.$$

Now take $Q' \equiv Q_0' \times Q_1'$ and $R_0' \times R_1'$, necessarily both idle. Clearly $P'\mathcal{S}Q'$; also extending one or other derivation as necessary using $R_i' \xrightarrow{1} R_i'$ (Proposition 7.2), we have

$$G\{\text{fix}\, XF/X\} \xrightarrow{a} \Rightarrow R'.$$

Moreover, $Q' \approx R'$ by Lemma 8.10.

(iii) $G \equiv \text{fix}\, YH$, $Y \not\equiv X$. In this case, the proof is entirely analogous to the corresponding case in Proposition 4.6. □

We may now deduce easily

Proposition 8.15. *$E \approx^{\cdot} F$ iff $E \approx^{c} F$.*

Proof. ($\Rightarrow$) If $E \approx^{\times} F$, then from Lemma 8.14 we have for any asynchronous context $\mathcal{C}[E] \approx^{\cdot} \mathcal{C}[F]$. By Lemma 8.13, it follows that $E \approx^{c} F$.

($\Leftarrow$) If $E \approx^{c} F$, then by taking all contexts of the form $[\ ] \times R$ we have, by definition, $E \times R \approx F \times R$. Thus $E \approx^{\cdot} F$ by definition. □

We now summarise the proof of our theorem.

Proof of Theorem 8.8. (1) $\Leftrightarrow$ (2) by Proposition 8.15, and (2) $\Leftrightarrow$ (3) by Lemma 8.11. $\approx^{c}$ is a congruence by Lemma 8.14 and Proposition 8.15, and the final property of the theorem follows from Proposition 8.15 by the definition of $\approx^{\cdot}$. □

Finally, a simple corollary which was true in CCS and is very useful in proof, is that guarding by an action makes equivalent agents congruent:

Corollary 8.16. *If $E \approx F$ then $a.E \approx^{c} a.F$.*

9. A particulate calculus

In CCS, two particulate actions can never occur simultaneously unless they are inverses; the rules of action demand that they be 'interleaved'. In ASCCS, by contrast, the action operator, e.g. in $\alpha\beta.P$, may require them to occur simultaneously. Even if we restrict ourselves to particulate action operators, then simultaneous occurrence of α and β is enforced in $\alpha.P \times \beta.Q$, and permitted in $\delta(\alpha.P) \times \beta.Q$.

Our first step in this section is to present a subcalculus of ASCCS in which, whenever particles may occur simultaneously, they may also occur in arbitrary sequences. Having established this property, we proceed to show that the subcalculus is in a sense isomorphic to CCS. To be precise: although its rules of action differ from those of CCS, it turns out that when we limit ourselves to observers who can only observe particles (thus denying them the power to observe simultaneity) then the observation equivalence relation for ASCCS becomes exactly the observation equivalence of CCS. This justifies the interleaving enforced by the action rules of CCS. (These rules have the advantage that the only actions to be considered are $\Lambda \cup \{1\}$, not the whole action group, and in consequence the size of expressions generated by the Expansion Theorem [8] is not so great.)

In this section we are therefore assuming that Act is freely generated by names Σ. Let us call $\Lambda \cup \{1\}$ the *simple* actions. (Recall $\Lambda = \Sigma \cup \bar{\Sigma}$ from Section 5.)

The first limitation of our subcalculus is to permit only simple action operators $\mu.$, $\mu \in \Lambda \cup \{1\}$. In this section we use λ to range over Λ and μ to range over $\Lambda \cup \{1\}$, as we did in [8].

Second, we must ensure that morphisms preserve simple action; these *simple* morphisms satisfy the condition $\phi(\Lambda) \subseteq \Lambda \cup \{1\}$.

Third, since product can enforce non-particulate actions, we replace it by the combinator $|$, called *composition*, defined by

$$P|Q \;=\; P \times \delta Q + \delta P \times Q.$$

This operator—which was taken as primitive in CCS—enjoys the following simple properties:

Proposition 9.1. $\delta(P|Q) \sim \delta P|Q \sim P|\delta Q \sim \delta P \times \delta Q$.

Proof. Use Proposition 5.5. □

Finally, we permit only restrictions of the form $\backslash N$, $N \subseteq \Lambda$ (Section 5).

Thus the language of our subcalculus is

$$E ::= X \mid \mu.E \mid \sum \tilde{E} \mid E|E \mid E\backslash N \mid \mathrm{fix}_i \tilde{X}\tilde{E} \mid E[\phi]$$

where $N \subseteq \Lambda$ and ϕ is simple. We choose to omit δE, as we wish the correspondence with CCS to be as close as possible. We shall now call this language CCS, despite the fact that infinite summation is admitted; indeed, the latter allows one to derive

all CCS constructs concerning data values and value-passing, as indicated in Section 6.

Let us denote by $\mathscr{P}^1$ and $\mathscr{E}^1$ the agents and expressions of CCS. To avoid confusion, for the rest of this section we shall use P and Q to stand for members of $\mathscr{P}^{AS}$ and $\mathscr{P}^1$ respectively. It is important to note that if Q is in $\mathscr{P}^1$ and has a derivative P', then P' is not necessarily in $\mathscr{P}^1$. But it is nearly so:

Proposition 9.2. *If $Q \in \mathscr{P}^1$ has a proper derivative P', then $P' \sim \delta Q'$ for some $Q' \mathscr{P}^1$.*

Proof. First show, by induction on inference, that if $Q \xrightarrow{a} P'$ then $P' \sim \delta Q'$ for some Q' in $\mathscr{P}^1$, using Proposition 9.1. Then the result is easy by induction on length of derivation of P'. □

We now wish to show that any derivative P of $Q \in \mathscr{P}^1$, which can perform a non-particulate action, can also perform the particles in any order; this means that CCS cannot enforce simultaneity.

Definition. Particles λ, λ' are *non-opposing* if $\lambda' \neq \bar{\lambda}$. Actions a, a' are *non-opposing* if Part(a), Part (a') are pairwise non-opposing.

Clearly any action is a unique product (up to order) of non-opposing particles, not necessarily distinct.

Proposition 9.3. *If P is a derivative of $Q \in \mathscr{P}^1$ and $P \xrightarrow{a} P'$ where $a = \lambda_1 \cdots \lambda_n$ is a product of non-opposing particles, $n \geq 1$, then, for some $P'' \sim P'$,*

$$P \xrightarrow{\lambda_1} \cdots \xrightarrow{\lambda_n} P''.$$

The proof is via the two following lemmas, whose proofs we omit.

Lemma 9.4. *Let $a = a_1 \cdots a_n$ be a product of non-opposing actions, and let $a = bc$. Then there are products $b = b_1 \cdots b_n$ and $c = c_1 \cdots c_n$ of non-opposing actions such that $a_i = b_i c_i$, $1 \leq i \leq n$.*

Lemma 9.5. *Let $Q \in \mathscr{P}^1$ and $Q \xrightarrow{a} P'$, where $a = a_1 \cdots a_n$ is a product of non-opposing actions with $a_1 \neq 1$. Then*

$$Q \xrightarrow{a_1} \cdots \xrightarrow{a_n} P'.$$

Proof. By induction on inference and Lemma 9.4. □

Proposition 9.3 then follows by Proposition 9.2, using a special case of the last lemma.

We now turn to the question of observation equivalence in CCS. Note first that, in contrast to the treatment in [8], the two CCS agents

$$\lambda_1.\mathbf{0}|\lambda_2.\mathbf{0}, \qquad \lambda_1.\lambda_2.\mathbf{0}+\lambda_2.\lambda_1.\mathbf{0}$$

are not equivalent in the sense of Section 8, since the first admits observation of the non-particulate action $\lambda_1\lambda_2$. But if such an observation is not in the observer's power they should be indistinguishable. To this end, let us modify the definitions of weak bisimulation and weak equivalence in Section 8 by restricting consideration to sequences $u \in \Lambda^*$ of *particulate* actions; we shall call this Λ-bisimulation, and denote the resulting equivalence by $\approx_\Lambda$.

On the other hand, we obtain a weak equivalence for CCS independently, treating all its operators as primitive, in the following way. We introduce action relations $\overset{\mu}{\twoheadrightarrow}$, $\mu \in \Lambda \cup \{1\}$, and their derivation rules are the same as for $\overset{\mu}{\rightarrow}$ in SCCS except for the CCS operations '$\mu.$' and '$|$' (which are derived operators in SCCS); for these, we have rules as in [8]:

Action: $\boxed{\mu.E \overset{\mu}{\twoheadrightarrow} E}$

Composition: (1) $\boxed{\dfrac{E \overset{\mu}{\twoheadrightarrow} E'}{E|F \overset{\mu}{\twoheadrightarrow} E'|F}}$ (2) $\boxed{\dfrac{F \overset{\mu}{\twoheadrightarrow} F'}{E|F \overset{\mu}{\twoheadrightarrow} E|F'}}$

(3) $\boxed{\dfrac{E \overset{\lambda}{\twoheadrightarrow} E' \quad F \overset{\bar{\lambda}}{\twoheadrightarrow} F'}{E|F \overset{1}{\twoheadrightarrow} E'|F'}}$

Note that the Action rule contrasts with $\mu.E \overset{\mu}{\rightarrow} \delta E$ in ASCCS, and the Composition rules preclude simultaneous occurrence of non-opposing particles.

Proceeding as in Section 8, we define relations $\overset{u}{\Rightarrow}$ for $u \in \Lambda^*$;

Definition. Let $u = \langle \lambda_1, \lambda_2, \ldots \lambda_n \rangle \in \Lambda^*$, $n \geq 0$. Then

$$Q \overset{u}{\Rightarrow} Q' \quad \text{iff} \quad Q(\overset{1}{\twoheadrightarrow})^* \overset{\lambda_1}{\longrightarrow} (\overset{1}{\twoheadrightarrow})^* \cdots \overset{\lambda_n}{\longrightarrow} (\overset{1}{\twoheadrightarrow})^* Q'.$$

Then, in terms of the corresponding notion of weak particulate bisimulation, which we shall call Π-bisimulation, we obtain an equivalence $\approx_\Pi$ over CCS, the maximum Π-bisimulation. The remainder of this section is devoted to an outline of the proof that $\approx_\Lambda$, restricted to CCS, is identical with $\approx_\Pi$. This theorem follows easily from a lemma whose proof we give later.

Lemma 9.6. *Let $Q \in \mathcal{P}^\setminus$, and $u \in \Lambda^*$.*

(1) *If $Q \overset{u}{\Rightarrow} P'$ then, for some $Q' \in \mathcal{P}^\setminus$, $Q \overset{u}{\twoheadrightarrow} Q'$ and $P' \sim \delta Q'$.*

(2) *If $Q \overset{u}{\twoheadrightarrow} Q'$ then, for some $P' \in \mathcal{P}^{\wedge S}$, $Q \overset{u}{\Rightarrow} P'$ and $P' \sim \delta Q'$.*

Theorem 9.7. *Let* $Q_1, Q_2 \in \mathscr{P}^{\Lambda}$. *Then*

$$Q_1 \approx_{\Lambda} Q_2 \quad \textit{iff} \quad Q_1 \approx_{\Pi} Q_2.$$

Proof. ($\Rightarrow$) It is enough to show that $\approx_{\Lambda}$ restricted to $\mathscr{P}^{\Lambda}$ is a Π-bisimulation. By Lemma 9.6, together with the fact that $P' \sim \delta Q'$ implies $P' \approx Q'$ and hence also $P' \approx_{\Lambda} Q'$, we can easily show that if $Q_1 \approx_{\Lambda} Q_2$ and $Q_1 \overset{\mu}{\Rightarrow} Q_1'$ then for some Q_2', $Q_2 \overset{\mu}{\Rightarrow} Q_2' \approx_{\Lambda} Q_1'$.

($\Leftarrow$) It is enough to show that $\approx_{\Pi}$ is a Λ-bisimulation up to $\approx_{\Lambda}$; the argument is similar to the first part. □

To prove Lemma 9.6, we need results relating the two kinds of single action $\overset{\mu}{\rightarrow}$ and $\overset{\mu}{\twoheadrightarrow}$. In one direction it is easy, since a single action $\overset{\mu}{\twoheadrightarrow}$ can always be matched by a single action $\overset{\mu}{\rightarrow}$.

Lemma 9.8. *If* $Q \overset{\mu}{\twoheadrightarrow} Q'$ *then, for some* P', $Q \overset{\mu}{\rightarrow} P' \sim \delta Q'$.

Proof. By induction on inference. In the case $Q \equiv Q_1 | Q_2$ Proposition 9.1 is needed. □

Proof of Lemma 9.6(2). From Lemma 9.8, it is easy to show by induction on n that if $Q \overset{\mu_1}{\twoheadrightarrow} \cdots \overset{\mu_n}{\twoheadrightarrow} Q'$ then, for some P', $Q \overset{\mu_1}{\rightarrow} \cdots \overset{\mu_n}{\rightarrow} P' \sim \delta Q'$. Now, recalling that $\mu_i \in \Lambda \cup \{1\}$ and using the definitions of $\overset{\mu}{\Rightarrow}$ and $\overset{\mu}{\Rightarrow}$, Lemma 9.6(2) is immediate. □

In the other direction, a single action $\overset{\mu}{\rightarrow}$ may be matched only by a *sequence of* actions $(\overset{1}{\twoheadrightarrow})^m \overset{\mu}{\twoheadrightarrow} (\overset{1}{\twoheadrightarrow})^n$, because of the greater degree of simultaneity allowed in ASCCS.

Lemma 9.9. *If* $a = \lambda_1 \cdots \lambda_n$ *is a product of non-opposing particles,* $n \geq 0$, *and* $Q \overset{a}{\rightarrow} P'$, *then, for some* Q',

$$Q (\overset{1}{\twoheadrightarrow})^* \overset{\lambda_1}{\twoheadrightarrow} (\overset{1}{\twoheadrightarrow})^* \cdots \overset{\lambda_n}{\twoheadrightarrow} (\overset{1}{\twoheadrightarrow})^* Q' \quad \textit{and} \quad P' \sim \delta Q'.$$

Proof. By induction on inference. Two cases are of interest:

(i) $Q \equiv \mu.Q_1$. Then *either* $\mu = a = 1$ and $n = 0$ *or* $\mu = a \in \Lambda$ and $n = 1$; in either case $P' \equiv \delta Q_1$ and $Q \overset{\mu}{\rightarrow} Q_1$.

(ii) $Q \equiv Q_1 | Q_2 \equiv Q_1 \times \delta Q_2 + \delta Q_1 \times Q_2$. Then by shorter inferences we have, without loss of generality,

$$Q_1 \overset{b}{\rightarrow} P_1', \quad \delta Q_2 \overset{c}{\rightarrow} P_2', \quad P' \equiv P_1' \times P_2' \quad \text{and} \quad a = bc.$$

There are now two subcases. If $c = 1$ and $P_2' \equiv \delta Q_2$, then $b = a$ and we use the induction hypothesis for Q_1 to obtain an action sequence ending with Q_1'; then

the required action sequence for Q is obtained entirely by use of derivation rule (1) for composition, and at the end $P' \equiv P_1' \times \delta Q_2 \sim \delta Q_1' \times \delta Q_2 \sim \delta(Q_1' | Q_2)$ by Proposition 9.1.

In the other subcase, $Q_2 \xrightarrow{c} P_2'$. We can find non-opposing products of particles $b = \lambda_{i_1} \cdots \lambda_{i_p} \pi_1 \cdots \pi_m$ and $c = \lambda_{j_1} \cdots \lambda_{j_q} \bar{\pi}_1 \cdots \bar{\pi}_m$ where the λ sequences merge to form $\lambda_1 \cdots \lambda_n$. We apply the induction hypothesis to obtain action sequences for Q_1 and Q_2, ending with Q_1' and Q_2', and all three derivation rules for composition then yield the required action section for Q, ending with $Q_1' | Q_2'$; the third composition rule is only needed for each pair π_k, $\bar{\pi}_k$.

We also have that $P' \equiv P_1' \times P_2' \sim \delta Q_1' \times \delta Q_2' \sim \delta(Q_1' | Q_2')$ by Proposition 9.1. □

Proof of Lemma 9.6(1). As special cases of Lemma 9.9 we have that $Q \xrightarrow{\lambda} P'$ implies $Q \overset{\lambda}{\Rightarrow} Q'$, and $Q \xrightarrow{1} P'$ implies $Q \Rightarrow Q'$, with $P' \sim \delta Q'$ in each case. Lemma 9.6(1) is then easily proved by induction on the length of the action sequence represented by $Q \overset{u}{\Rightarrow} P'$. □

This completes the proof of Theorem 9.7. □

To conclude this section, we state without proof a characterization of the weak equivalence $\approx_{II}$ over CCS, for comparison with the results of Section 8. In fact, the results are exactly those obtained in [8], though we must stress that the present equivalence $\approx_{II}$ is slightly different from that of [8]—it is the maximum fixed point of a mapping $\mathcal{G}_{II}$ of relations, while in [8] we took the infinite intersection $\bigcap_{n \geq 0} \mathcal{G}_{II}^n(\mathcal{U})$, $\mathcal{U}$ being the universal relation.

As in Section 8, $\approx_{II}$ is not preserved by summation. Taking $P \equiv \lambda.\mathbf{0}$, $Q \equiv 1.\lambda.\mathbf{0}$, $R \equiv \lambda'.\mathbf{0}$ we have $P \approx_{II} Q$ but $P + R \not\approx_{II} Q + R$. However, in contrast to Section 8, all other operators of CCS preserve $\approx_{II}$.

Proposition 9.10. *If* $Q_1 \approx_{II} Q_2$ *then* $\mu.Q_1 \approx_{II} \mu.Q_2$, $Q_1 | Q_3 \approx_{II} Q_2 | Q_3$, $Q_1 \backslash M \approx_{II} Q_2 \backslash M$ *and* $Q_1[\phi] \approx_{II} Q_2[\phi]$.

As before, we expect the largest congruence contained in $\approx_{II}$ to be $=_{II}$, defined by

$$Q_1 =_{II} Q_2 \quad \text{iff for all CCS contexts } C[\ ],\quad C[Q_1] \approx_{II} C[Q_2].$$

In fact, to compare with Theorem 8.8 we can show that summation contexts are the critical ones:

Theorem 9.11. *The following are equivalent in* CCS:

(1) $Q_1 =_{II} Q_2$.

(2) *For all* Q_3 *in* CCS, $Q_1 + Q_3 \approx_{II} Q_2 + Q_3$.

(3) *For all* $\mu \in \Lambda \cup \{1\}$,

(i) *if* $Q_1 \xrightarrow{\mu} Q_1'$ *then, for some* Q_2', $Q_2 \overset{\mu}{\Rightarrow} Q_2'$ *and* $Q_1' \approx_{II} Q_2'$,

(ii) *if* $Q_2 \xrightarrow{\mu} Q_2'$ *then, for some* Q_1', $Q_1 \overset{\mu}{\Rightarrow} Q_1'$ *and* $Q_1' \approx_{II} Q_2'$.

Moreover, $\approx_{\Pi}^{c}$ *is a congruence and, if* $\tilde{X} = \mathrm{Free}(E, F)$, *then*

$$E \approx_{\Pi}^{c} F \quad \textit{iff for all agents } \tilde{Q} \quad E\{\tilde{Q}/\tilde{X}\} \approx_{\Pi}^{c} F\{\tilde{Q}/\tilde{X}\}.$$

Note that clause (3) imposes a slightly stronger condition than Π-bisimulation, only upon initial 1-actions of Q_1 and Q_2.

By analogy with Corollary 8.16, we also have

Corollary 9.12. *If* $E \approx F$ *then* $\mu.E \approx^{c} \mu.F$.

10. Discussion

The results of Part II relate the synchrony of SCCS and the asynchrony of CCS in perhaps the simplest way that we could expect. It was seen that the observation congruence depends both on how much can be observed (all actions except 1 in ASCCS, or just particles in CCS) and on the operators admitted (product in ASCCS, or composition in CCS); this indicates that matters are quite delicate. But difficult questions remain. In particular, we have not said anything about synchronous or timed agents in asynchronous contexts. As asynchronous context $\mathscr{C}[\ ]$ may have the property that for all *asynchronous* P and Q, $P \approx Q$ implies $\mathscr{C}[P] \approx \mathscr{C}[Q]$, i.e. $\mathscr{C}[\ ]$ preserves equivalence in ASCCS, yet it does *not* follow that $\mathscr{C}[\ ]$ has this property when P and Q are *synchronous*. But we would like to know for which contexts this *does* hold; then we should be able to build a synchronous agent P knowing that, in such contexts, its behaviour up to weak equivalence is all that matters. K. Mitchell at Edinburgh has studied this and obtained some preliminary results.

Concerning fixed-points, Proposition 4.9 gave a satisfactory condition under which they are unique in SCCS up to strong congruence. Uniqueness up to weak congruence, in the asynchronous case, is not so straightforward. For CCS, Sanderson has encouraging results [18]; his sufficient conditions for uniqueness will be adequate for many practical situations. A more complete understanding—for ASCCS too—is needed, bringing in also the order-theoretic approach. Hennessy and Plotkin [5] studied a pre-order relation over CCS agents, which yield naturally the existence of least fixed points; what is unclear is whether other ordering relations are as good or better, and how they relate to the present approach in which order has been ignored. The order-theoretic approach in [2] is completely different, and corresponds to a more generous equivalence than our $\approx$, i.e. one in which more agents are (intentionally) equated.

Our treatment of asynchrony, so far, does not model the related notions of liveness and fairness. By the delay operator δ we can model an agent which will wait indefinitely, even for ever, to perform its task. This is suitable for many purposes: data-structures which may or may not be accessed, or programs which

may or may not be entered. But typically an asynchronous system contains also *live* agents which, at a suitable abstract level of modelling, can be considered to wait for arbitrary finite time but not infinitely. Such systems are rather well treated by the temporal logics of Lamport, Pnueli and others [6, 17]; it remains to refine the present algebraic approach to deal with them. Preliminary studies indicate that a new operator ε (connoting 'expectation' or 'eventually') may be introduced, as an active analogue of δ but—unlike δ—not derivable. The notion of bisimulation also seems to need refinement. If this can be done, then we may be able to claim that *every* aspect of asynchrony can be treated in SCCS. More generally, it is also important for system verification to make connections between algebraic calculi and forms of temporal logic.

Although SCCS, by allowing arbitrary products of actions, reflects concurrency more than its predecessor CCS, it does not reflect concurrency in the sense of causal independence, as does Petri's Net Theory. For example, using the derived action and composition operators in SCCS, we have

$$a.\mathbf{0}|b.\mathbf{0} \sim ab.\mathbf{0}+a.b.\mathbf{0}+b.a.\mathbf{0}$$

while in Net Theory the equation would not be accepted, since the actions a and b would be regarded as causally independent in the left-hand agent but not so in the right-hand agent. Our reason for accepting the equation is that an observer can detect sequence and simultaneity, but not causation. However, a more intensional model which distinguishes the above agents will be of analytical value (further evidence that there are many candidates for the notion of *process*, since there are many useful congruence relations over agents). The Event Structures studied by Nielsen, Plotkin and Winskel [14, 19] provide such a model, and also form a bridge between Net Theory and the present work.

In summary, we have presented a calculus with four combinators and a construct for recursion, and by various derived constructions we have shown that it is widely expressive. Based upon the operational rules we have also built various interpretations (via congruences) of the calculus; these are motivated by simple intuitions and yield useful mathematical properties, but we have also seen that there is considerable freedom in the choice of interpretation. We hope that further work will classify the interpretations, but that the set of basic combinators hardly needs to be extended.

Appendix. Definability of agent morphism

For any morphism ϕ: Act → Act, the actions of $E[\phi]$ are characterized as follows:

(1) if $E \xrightarrow{a} E'$, then $E[\phi] \xrightarrow{\phi(a)} E'[\phi]$,

(2) if $E[\phi] \xrightarrow{b} G'$, then $E \xrightarrow{a} E'$ for some a and E' such that $b = \phi(a)$ and $G' = E'[\phi]$.

Therefore we must find a derived operator $[\phi]$ satisfying these two properties.

For this purpose we make the assumption that Act is a group, and that it is arbitrarily extendible to $\mathrm{Act} \otimes \mathrm{Act}^{\dagger}$ (the direct product of Abelian groups) where $(\mathrm{Act}^{\dagger}, \times, \bar{}, 1)$ is again an Abelian group. The operations in the product group are

$$(a, a') \times (b, b') = (a \times b, a' \times b'),$$

$$\overline{(a, a')} = (\bar{a}, \bar{a}'), \qquad 1 = (1, 1)$$

and $\mathrm{Act} \simeq \mathrm{Act} \otimes \{1\}$, $\mathrm{Act}^{\dagger} \simeq \{1\} \otimes \mathrm{Act}^{\dagger}$ are subgroups of the product group.

In particular, choose $\mathrm{Act}^{\dagger}$ isomorphic to Act, under the isomorphism $a (\in \mathrm{Act}) \mapsto a^{\dagger} (\in \mathrm{Act}^{\dagger})$. Our first step in defining $[\phi]$ is to lift this isomorphism to agent expressions by defining

$$E^{\dagger} \equiv \left(E \times \mathrm{fix}\, X\left(\sum_{a \in \mathrm{Act}} \bar{a} a^{\dagger} : X\right)\right) \upharpoonright \mathrm{Act}^{\dagger}.$$

The second factor of the product simply converts every action $a \in \mathrm{Act}$ of E to the corresponding action $a^{\dagger} \in \mathrm{Act}^{\dagger}$ of $E^{\dagger}$, and the restriction ensures that every action of $E^{\dagger}$ must be so derived. More precisely, we can easily prove the following

Lemma A.1. (1) *If* $E \xrightarrow{a} E'$, *then* $E^{\dagger} \xrightarrow{a^{\dagger}} E'^{\dagger}$.

(2) *If* $E^{\dagger} \xrightarrow{a^{\dagger}} F'$, *then* $E \xrightarrow{a} E'$ *for some* E' *such that* $F' \equiv E'^{\dagger}$.

The second step is similar, but involves the given morphism $\phi : \mathrm{Act} \to \mathrm{Act}$. For arbitrary F with actions in $\mathrm{Act}^{\dagger}$, define

$$F^{\phi} \equiv \left(F \times \mathrm{fix}\, X\left(\sum_{a \in \mathrm{Act}} \bar{a}^{\dagger} \phi(a) : X\right)\right) \upharpoonright \mathrm{Act}$$

and we can prove in a similar way

Lemma A.2. (1) *If* $F \xrightarrow{a^{\dagger}} F'$, *then* $F^{\phi} \xrightarrow{\phi(a)} F'^{\phi}$.

(2) *If* $F^{\phi} \xrightarrow{b} G'$, *then* $F \xrightarrow{a^{\dagger}} F'$ *for some* $a \in \mathrm{Act}$ *and some* F' *such that* $b = \phi(a)$ *and* $G' \equiv F'^{\phi}$.

To complete our definition we merely compose our two transformations:

$$E[\phi] \equiv (E^{\dagger})^{\phi}.$$

Then it is a direct consequence of the two lemmas that

Proposition A.3. (1) *If* $E \xrightarrow{a} E'$, *then* $E[\phi] \xrightarrow{\phi(a)} E'[\phi]$.

(2) *If* $E[\phi] \xrightarrow{b} G'$, *then* $E \xrightarrow{a} E'$ *for some* a *and* E' *such that* $b = \phi(a)$ *and* $G' \equiv E'[\phi]$.

It would be interesting to know if this construction can be done somehow without extending Act. The author has only been able to make this work for the case in which ϕ is idempotent.

References

[1] L. Cardelli (1982), Real time agents, *Proc. ICALP 82*, Lecture Notes in Computer Science **140** (Springer, Berlin, 1982) 94–106.

[2] C. Hoare, S. Brookes and W. Roscoe, A theory of communicating sequential proce[illegible], Programming Research Group Oxford University (1981).

[3] M. Hennessy, A term model for synchronous processes, CSR-77-81, Computer Science Department, Edinburgh University (1981).

[4] C. Hoare, A model for communicating sequential processes, Programming Research Group, Oxford University (1978).

[5] M. Hennessy and G. Plotkin, A term model for CCS, *Proc. 9th MFCS*, Poland, Lecture Notes in Computer Science **88** (Springer, Berlin, 1982).

[6] L. Lamport, 'Sometime' is sometimes 'not never', *Proc. 7th ACM Symposium on Principles of Programming Languages* (1980).

[7] R. Milner, Processes; a mathematical model of computing agents, in: H.E. Rose and J.C. Shepherdson, Eds., *Logic Colloquium* 72 (North-Holland, Amsterdam, 1973) 157–174.

[8] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).

[9] R. Milner, On relating synchrony and asynchrony, CSR-75-80, Computer Science Department, Edinburgh University (1981).

[10] R. Milner, A complete inference system for a class of regular behaviours, CSR-111-82, Computer Science Department, Edinburgh University (1982).

[11] R. Milner, Four combinators for concurrency, *Proc. ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1982, Ottawa (1982) 104–110.

[12] G. Milne, Synchronized behaviour algebras; a model for interacting systems, Department of Computer Science, University of Southern California (1979).

[13] G. Milne and R. Milner, Concurrent processes and their syntax, *J.ACM* **26** (2) (1979) 302–321.

[14] M. Nielson, G. Plotkin and G. Winskel, Petri nets, event structures and domains, Part I, *Theoret. Comput. Sci.* **13** (1) (1981) 85–108.

[15] D. Park, Concurrency and automata on infinite sequences, *Proc. 5th GI Conference*, Lecture Notes in Computer Science **104** (Springer, Berlin, 1981).

[16] G. Plotkin, A structured approach to operational semantics, DAIMI FN-19, Computer Science Department, Aarhus University (1981).

[17] A. Pnueli, The temporal semantics of concurrent programs *Theoret. Comput. Sci.* **13** (1) (1981) 45–60.

[18] M. Sanderson, Forthcoming Ph.D. Thesis, Computer Science Department, Edinburgh University (1982).

[19] G. Winskel, Event structure semantics for CCS and related languages, *Proc. ICALP* 82, Lecture Notes in Computer Science **140** (Springer, Berlin 1982) 561–576.