Action Structures and the Pi Calculus

Robin Milner

Laboratory for Foundations of Computer Science, Computer Science Department, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, UK[1]

Abstract: Action structures are proposed as a variety of algebra to underlie concrete models of concurrency and interaction. An action structure is equipped with *composition* and *product* of actions, together with an indexed family of *abstractors* to allow parametrisation of actions, and a *reaction* relation to represent activity. The eight axioms of an action structure make it an enriched strict monoidal category.

In Part I the notion of action structure is developed mathematically, and examples are studied ranging from the evaluation of expressions to the statics and dynamics of Petri nets. For algebraic process calculi in particular, it is shown how they may be defined by a uniform superposition of process structure upon an action structure specific to each calculus.

The theory of action structures emphasizes the notion of *effect*; that is, the effect which any interaction among processes exerts upon its participants. Effects together with *incidents* (roughly speaking, observable actions) allow a uniform treatment of bisimulation congruence.

In Part II, the π-calculus is treated as an extended example of this uniform process theory. Three action structures for the π-calculus are examined; they are presented in a simple graphical form. The incidents for π-calculus are best characterized in terms of *reachability*, a graphical notion. This leads to a bisimulation congruence for the *synchronous* π-calculus, in which individual transitions may involve actions of arbitrary complexity.

Keywords: abstractor, action, action structure, bisimulation, concurrency, effect, incident, interaction, monoidal category, process structure, pi-calculus, reaction relation, transition.

Contents:

[1]This work was done with the support of a Senior Fellowship from the Science and Engineering Research Council, UK.

PART I: Action Structures

1 Introduction

The purpose of this paper is to explore action structures; they provide a new algebraic foundation for concrete models of concurrent computation, but are free from the ad hoc details which are often present in such models.

1.1 Background As far as modelling sequential computation is concerned, a classic mathematical structure is Scott's notion of *domain* [13]. It allows us to understand complete computations by examining the partial computations which approximate them. Indeed domains also help us to understand concurrent computations in this way, as shown by Nielsen, Plotkin and Winskel [25] in an important paper which relates domains both to event structures and to Petri nets [27].

Event structures and Petri nets are good examples of mathematical structures specific to concurrency. But we appear still to lack a canonical structure which is *combinational*, i.e. which explains how processes are synthesized, and which embodies the dynamics of *interaction* among processes. There are algebraic process calculi [3, 14, 19] and algebraic theories of Petri nets [7, 9]; these models have theoretical content which emphasizes combination and interaction. But what they have in common has not been distilled, so that as a family –and even individually– they can appear ad hoc.

It is therefore reasonable to look for a canonical algebraic structure for concurrency, with interaction at its heart; in this paper we propose *action structures* as a candidate. An action structure is an algebra which allows complex actions to be built from simple ones; in addition it has a dynamic component which defines the possibly complex performance of an action.

Central to action structures are two concerns. The first is that an action may be simple in one view, complex in another. Here are two examples, to suggest that this is consistent with common usage of the term "action". First, think of a sequential program accessing the fourteenth member of a linked list. At a low level this consists of many actions –chaining down the list– and this is how an implementer must treat it; at another level it is clearly a single action, an interaction between a program and the memory. As another example, suppose that you treat me to lunch because you want to borrow money from me. The lunch consists of many interactions, undoubtedly a complex pattern! Later, however, you and I will remember it as a single action: you stung me. This relativity of atomicity is well known to systems engineers; one only has to think of the difficult questions associated with transactions in a database. Studies of action refinement in process algebras, e.g. by van Glabbeek and Goltz [12] and Aceto and Hennessy [1], have begun to place relative atomicity on a theoretical footing; one purpose of action structures is to continue the task.

The second concern of action structures is with the *effect* which a shared action exerts upon its participants. Each participant suffers a different effect. In the case of the lunch you are richer, I am stung, the waiter is tired. [2] For a more formal example consider the following transition in CCS [19]:

$$\overline{p}x.P \mid p(y).Q \;\longrightarrow\; P \mid \{x/y\}Q,$$

[2] The waiter also took part, and illustrates another difficulty with concurrency. Unlike you and me, he will not remember our lunch as an atomic action; for him, it was inextricably interwoven with the serving of other lunches. This shows that the level of atomicity does not only vary from one global view of a system to another; it even varies between parts of the system.

where p is a port at which x is being output by one process and at which the other is ready to receive input, and $\{^x\!/y\}Q$ means the substitution of x for y in Q. In action structures this will be modelled by a *reaction*, which may be written roughly as

$$\overline{p}x \otimes p(y) \searrow \mathbf{id} \otimes \{^x\!/y\}.$$

In general, a reaction $a \searrow a'$ in an action structure represents part of the performance of an action a; the performance may continue by further reaction $a' \searrow a''$, and so on. In the example, a reaction of the product of two actions (itself an action) reduces it to a remainder action which is the product of **id** and $\{^x\!/y\}$; these, and their product, are degenerate actions which we call *effects*. They are degenerate in that they perform no further reaction; instead they represent the effect exerted upon P and Q respectively by the performance of the original action, i.e. by the occurrence of a communication through the port p. Note that the unit action **id**, the effect on P, represents *no* effect; it is only Q that feels an effect, which in this case is a substitution.

Strong motivation for action structures came from the π-calculus [24]. A distinguishing feature of the π-calculus is that port-names, like p in the above, can be transmitted as data; this can create another kind of effect, namely a new liaison between two agents. To illustrate, in the π-calculus we may adapt the previous example as follows:

$$((\boldsymbol{\nu}x)\overline{p}x.P) \mid p(y).Q \longrightarrow (\boldsymbol{\nu}x)(P \mid \{^x\!/y\}Q).$$

Here the restriction $(\boldsymbol{\nu}x)$ localises the port x to $\overline{p}x.P$; but an effect of the reaction is to widen the scope of x, so that it becomes a port which can carry information between P and Q. In action structures the corresponding reaction is now roughly speaking

$$((\boldsymbol{\nu}x)\overline{p}x) \otimes p(y) \searrow (\boldsymbol{\nu}x)(\mathbf{id} \otimes \{^x\!/y\}),$$

and we see the newly created liaison as a component of the resulting effect. Thus the π-calculus suggested that one should try to distill a mathematical notion of effect from these instances.

The π-calculus also gave a second motivation. Through its treatment of names and restriction the π-calculus allows a complex action, such as accessing the n^{th} element of a list, to be programmed in terms of many simple actions; so one would like a version of the π-calculus in which such a complex action can be a single transition of a process. This would correspond to the original π-calculus in the way that Synchronous CCS [18], where a transition may involve arbitrarily many "particulate" interactions, corresponds to CCS where it may only involve a single one. This motivation is traced in Part II, Section 10; it leads naturally to the three algebraic operations of an action structure. A main outcome of this paper is that process calculi like CCS, the π-calculus and their variants can be factored into two parts: at the lower level an action structure which is different for each of them, and at the upper level a process structure which is common to them all.

1.2 Outline of Part I In Section 2 we introduce the definition of action structures. We first present static action structures as an algebraic theory; then we add dynamics, a preorder preserved by the algebraic operations; finally we define a natural notion of homomorphism and substructure.

Section 3 gives several examples of action structures: functions, expression evaluation, process calculi and Petri nets. It also suggests how to treat the notion of *port*, or communication interface, uniformly in action structures. These examples lie comfortably within the action structure framework.

In Section 4, we treat the central notion of *effect*, the influence of the performance of an action on its participants. Effects take their place as a sub-actionstructure consisting of *inert* actions – those which have nothing left to perform. Some of the examples of Section 3 are re-visited to see what effect structure is appropriate in each case.

The next four sections are devoted to building process structure in a uniform way upon an action structure. Sections 5 and 6 deal respectively with the statics and with the dynamics of processes. The latter comes from two sources: from actions, especially the communications between processes, and from the process control operators.

Sections 7 and 8 present an important consequence of the theory: that the natural bisimulation equivalence over processes, presented uniformly in terms of the underlying action structure, is indeed a congruence relation. This crucially depends upon the mathematical properties of effects, and also of a new kind of actions known as *incidents*. An incident-set is a subset of the actions satisfying a simple property. There are many possible incident-sets, and each one determines a bisimulation equivalence; the larger the incident-set, the finer the equivalence. Section 7 defines incident-sets, and proves that under a simple condition there is a smallest incident-set including a given set of actions. Section 8 establishes that every bisimulation equivalence based upon an incident-set is a congruence.

Section 9 alludes to work by other authors which is related, or with which a relation should be sought, apart from papers already mentioned in the text.

1.3 Acknowledgements I am grateful to Yoram Hirshfeld, Davide Sangiorgi and David Turner for useful discussions. The presentation has been strengthened by some category-theoretic ideas with the help of Barry Jay and John Power, whom I thank for their insights.

2 Action structures

An action structure is an algebra with some extra structure. It may also be seen as strict monoidal category with two additions, one structural and one dynamic. The elements of an action structure are called *actions*; they are its morphisms, in categorical terms. The structural addition is an indexed family of functions over actions called *abstractors*; categorically they are endo-functors. Abstractors allow actions to be parametric upon certain constituents. The dynamic addition is called *reaction*; that is, a preorder over actions which represents activity.

2.1 Definition (static action structure) Let X be a set; in this context its members $x, y, \ldots$ will often be called *names*. Then a *static action structure* A over X consists of

- A monoid $(M, \otimes, \epsilon)$ of *arities*, and an assignment to each x of an arity in M. Categorically, M is the monoid of objects.

- For each $m, n \in M$ a set $A_{m,n}$ of *actions*. If $a \in A_{m,n}$ we say that a has *arity* $m \to n$, and write $a : m \to n$.

- Nullary operators $\mathbf{id}_m$ (the identities), binary operators $\cdot$ (composition) and $\otimes$ (tensor product), and a family $\{\mathbf{ab}_x \mid x \in X\}$ of unary operators (abstractors).

The operators are subject to the following rules of arity:

$$\mathbf{id}_m : m \to m \qquad\qquad \frac{a : k \to n \qquad b : m \to p}{a \otimes b : k \otimes m \to n \otimes p}$$

$$\frac{a : k \to m \qquad b : m \to p}{a \cdot b : k \to p} \qquad\qquad \frac{x : k \qquad a : m \to n}{\mathbf{ab}_x a : k \otimes m \to k \otimes n}$$

and we freely omit the explicit arity m in $\mathbf{id}_m$, meaning that it can be supplied in any way which is consistent with the context. The operators satisfy eight laws:

C1.	$a \cdot \mathbf{id} = a = \mathbf{id} \cdot a$	C2.	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
P1.	$a \otimes \mathbf{id}_\epsilon = a = \mathbf{id}_\epsilon \otimes a$	P2.	$a \otimes (b \otimes c) = (a \otimes b) \otimes c$
PF1.	$\mathbf{id} \otimes \mathbf{id} = \mathbf{id}$	PF2.	$(a \cdot b) \otimes (c \cdot d) = (a \otimes c) \cdot (b \otimes d)$
AF1.	$\mathbf{ab}_x \mathbf{id} = \mathbf{id}$	AF2.	$\mathbf{ab}_x(a \cdot b) = (\mathbf{ab}_x a) \cdot (\mathbf{ab}_x b).$

■

Often the arity monoid M will be $(\mathsf{N}, +, 0)$, the natural numbers under addition; then the arity of each name x will be 1. But we may instead wish the arity of a name to be a *sort*, in some set $\mathcal{S}$; then M will be the free monoid generated by $\mathcal{S}$. More complex arities also arise in constructions of action structures.

The laws not involving the abstractors, i.e. C, P and PF, are just those of a strict monoidal category. PF asserts that $\otimes$ is a (bi)functor. Similarly, AF asserts that each $\mathbf{ab}_x$ is a functor.

We now proceed to the dynamics.

2.2 Definition (dynamic action structure) A (*dynamic*) *action structure* A over X consists of a static action structure together with a preorder $\searrow$ over each $A_{m,n}$ called *reaction*, such that

- Each operator preserves reaction, e.g. $a \searrow a'$ implies $a \cdot b \searrow a' \cdot b$ and $b \cdot a \searrow b \cdot a'$;
- $\mathbf{id}$ is inactive, i.e. $\mathbf{id} \searrow a$ implies $a = \mathbf{id}$. ■

Reaction is often the transitive reflexive closure of an elementary relation $\overset{1}{\searrow}$ which may be called *atomic reaction*, representing the occurrence of an elementary event. An example is the interaction between complementary actions $\alpha, \overline{\alpha}$ in CCS; in Section 3.6 we shall see that $\alpha \otimes \overline{\alpha} \overset{1}{\searrow} \tau$.

One of the aims in studying action structures is to learn how to treat the *composite* reactions in an action structure as the *elementary* steps of a higher-level process calculus. In Sections 5 and 6 we show how to build familiar process calculi in a uniform way upon an action structure; then the transition relation, already familiar in process calculi, is definable in terms of the lower-level reaction relation.

2.3 Terminology We shall usually consider dynamic, not static, action structures; so we usually omit the adjective "dynamic".

When the arity monoid M and the name-set X are understood we shall just use A to denote either the action-set or the entire action structure. At other times we may write (M, X, A) and (N, Y, B) for two distinct action structures; we may write $(M, X, A, \searrow)$ if we want to emphasise reaction; we may also write M_A, X_A, $\searrow_A$ to denote the parts of a given action structure A. Such abuses, used with care, will save heavy formality.

2.4 Definition (homomorphism of action structures) A *homomorphism* $\Phi : A \to B$ between two action structures A and B consists of

- A monoid homomorphism $\Phi : M_A \to M_B$;
- A map $\Phi : X_A \to X_B$ such that $x : m$ implies $\Phi x : \Phi m$;
- A map $\Phi : A \to B$ of actions such that

  $a : m \to n$ implies $\Phi a : \Phi m \to \Phi n$;
  Φ preserves all the operations **id**, $\cdot$, $\otimes$ and $\mathbf{ab}_x$;
  $a \searrow_A a'$ implies $\Phi a \searrow_B \Phi a'$.

Φ is an *isomorphism* if there is a homomorphism $\Phi^{-1} : B \to A$ such that both $\Phi \circ \Phi^{-1}$ and $\Phi^{-1} \circ \Phi$ are identity maps on arities, names and actions. ■

As we add further structure to the notion of action structure, we may also refine the notion of homomorphism.

2.5 Definition (sub-actionstructure) An action structure B is a *sub-actionstructure* of A if there is a homomorphism $\Phi : B \to A$ such that

1. Φ is injective on X_B, M_B and B ;
2. For all $b, b' \in B$, $b \searrow_B b'$ iff $\Phi b \searrow_A \Phi b'$. ■

In defining substructure notions it is common to require the implication in clause 2 only from left to right, and to use the term *full* for a substructure obeying the double implication. But for the present we have no use for the weaker condition, so we adopt the stronger one to avoid tediously repeating the term “full”.

2.6 Discussion The aspect of action structures which is least familiar is abstraction. The examples of Section 3 below will show that abstraction permits parametrisation of actions upon entities of some kind; but it also shows that these entities can be quite diverse in nature: ordinary values in one case, port-names in another, places and transitions in the case of Petri nets. So the notion of action structure presents, via abstraction and its axioms, something common to all these parametrisations.

But does it capture *all* that they have in common? This is really a vague question, since we clearly don’t want to limit ourselves to the examples of Section 3. Certainly for those applications it will be seen that the names X stand for entities, and that the abstractors bind those names. In fact for each name x there is usually an action $\langle x \rangle$ such that $\langle x \rangle \cdot \mathbf{ab}_y a$ imposes the substitution $\{x/y\}$ upon a; to be precise, it represents $\langle x \rangle \otimes \{x/y\}a$.

We might therefore expect concrete notions such as *free name*, *substitution* and *alpha-conversion* to be captured by the formulation and axioms of action structures. But they are not, for two reasons. First, the present axioms of action structure lie easily in the mind because they have a simple and appealing pattern. Second, they are strong enough to deliver interesting properties of process calculi in a uniform way, which is a main purpose this paper. However, these concrete notions are axiomatized for a subclass of action structures called *action calculi*, which are introduced in [21].

2.7 Categorical formulation Having discussed how the notion of action structure may be *strengthened*, we now look briefly at the natural formulation in categorical terms, which turns out to be a little *weaker* (i.e. more general).

The axioms AF require the abstractors to be functors. But we have not allowed the functors $\mathbf{ab}_x$ to be quite arbitrary; the rule of arity for $\mathbf{ab}_x a$ in Definition 2.1 asserts that the operation of the functor $\mathbf{ab}_x$ upon objects is taken to be $k \otimes (-)$, where k is fixed by x. But this relationship between the abstraction functors and the monoidal operation is not essential for many properties of action structures. Thus a more abstract definition of action structure would avoid specifying how the arity $\mathbf{ab}_x m$ depends upon x. The more specific formulation of Definition 2.1 aims at a comfortable compromise between mathematical generality and the specific interesting applications.

Concerning notation: I have adopted the forward version of composition, so that for $a : k \to m$ and $b : m \to p$ I write $a \cdot b$, not $b \circ a$. It would be confusing to do otherwise in this work, since in Sections 5 and 6 we find that composition $a \cdot P$ is closely allied to the prefixing operation of $a \bullet P$ in process algebra. Some prefer to use ";" for forward composition, but the use of punctuation in an algebraic expression can be distracting.

3 Examples of action structures

We begin by fitting some widely-known ideas into the framework of action structures, and then proceed to show how these structures can accommodate less standard notions found in concurrency theory.

3.1 Functions Let D be some fixed set of values. Define $\mathsf{FUN} = (M, X, A, \searrow)$ to be the action structure in which

$$\begin{array}{rcl} M &=& (\mathsf{N}, +, 0) \\ X &=& \emptyset \\ A_{m,n} &=& D^m \to D^n \\ \searrow &=& \text{the identity relation.} \end{array}$$

The operations over A are as follows, where we use $\vec{v}$ for a tuple $v_1 \dots v_m$ of variables, and sometimes use a comma to concatenate tuples:

$$\begin{array}{rcl} \mathbf{id}_m &\stackrel{\text{def}}{=}& \lambda\vec{v}\,.\,\vec{v} \\ f \cdot g &\stackrel{\text{def}}{=}& \lambda\vec{v}\,.\,g(f\vec{v}) \\ f \otimes g &\stackrel{\text{def}}{=}& \lambda\vec{u}\vec{v}\,.\,(f\vec{u}, g\vec{v}) \qquad (\vec{u}, \vec{v} \text{ all distinct}). \end{array}$$

There are no abstractors, so we only have to verify the axioms of a monoidal category; this is simple and well-known.

3.2 Functions with abstractors Let X be a set, which we shall call the *global variables*, and let $\text{Env} \stackrel{\text{def}}{=} X \to D$, the valuations of global variables in D. Define the action structure $\mathsf{FUN}[X] = (M, X, A, \searrow)$ as follows:

$$M = (\mathsf{N}, +, 0)$$

$$\begin{array}{rcl} X & = & \text{as given} \\ A_{m,n} & = & \text{Env} \to (D^m \to D^n) \\ \searrow & = & \text{the identity relation.} \end{array}$$

The operations are nearly as simple as in FUN; we only have to distribute the parameter $\varrho \in$ Env properly. Let us write $\varrho_{x\mapsto v}$ for the valuation which maps x to the value v, but otherwise assigns values as ϱ does. Then

$$\begin{array}{rcll} \mathbf{id}_m \varrho & \stackrel{\text{def}}{=} & \lambda\vec{v}\,.\,\vec{v} & \\ (f \cdot g)\varrho & \stackrel{\text{def}}{=} & \lambda\vec{v}\,.\,g\varrho(f\varrho\vec{v}) & \\ (f \otimes g)\varrho & \stackrel{\text{def}}{=} & \lambda\vec{u}\vec{v}\,.\,(f\varrho\vec{u}, g\varrho\vec{v}) & \quad (\vec{u}, \vec{v} \text{ all distinct}) \\ (\mathbf{ab}_x f)\varrho & \stackrel{\text{def}}{=} & \lambda u\vec{v}\,.\,(u, f\varrho_{x\mapsto u}\vec{v}) & \quad (u, \vec{v} \text{ all distinct}). \end{array}$$

Thus $(\mathbf{ab}_x f)\varrho$ takes one more argument than $f\varrho$; besides pushing this argument (u) through as a result, it also treats it as the value of x when evaluating f. It is instructive to check the laws AF for abstractors (2.1).

Clearly FUN is isomorphic to $\mathsf{FUN}[\emptyset]$. More generally, whatever X is, there is an injective homomorphism from FUN to $\mathsf{FUN}[X]$; it sends each $f \in D^m \to D^n$ to the function $g \in \text{Env} \to (D^m \to D^n)$ such that $g\varrho\vec{v} = f\vec{v}$ for every $\varrho \in$ Env. Thus FUN is a sub-actionstructure of $\mathsf{FUN}[X]$.

3.3 Evaluating expressions We have seen how functions constitute a static action structure. We now wish to consider how the expressions which denote functions constitute a *dynamic* action structure, which is given meaning by a homomorphism to $\mathsf{FUN}[X]$. It must be repeated that we are doing little more than casting familiar ideas in a new form. In effect we are presenting a dynamic variant of a Lawvere algebraic theory, with equations replaced by reactions; see Barr and Wells [4].

Let Exp_m be the class of expressions built from the global variables X and formal (or local) variables $y_1, \ldots, y_m$, by a fixed set of operators standing for operations over D. Thus Exp_m is the smallest set such that

- Each of $y_1, \ldots, y_m$ and each $x \in X$ is an expression in Exp_m;
- If $e_1, \ldots, e_k$ are expressions in Exp_m and $\underline{op}$ is a k-ary operator (standing for a function $op \in D^k \to D$), then $\underline{op}\langle e_1, \ldots, e_k\rangle$ is an expression in Exp_m.

An expression in Exp_m obviously denotes a function in $\text{Env} \to (D^m \to D)$, i.e. an action with arity $m \to 1$ in $\mathsf{FUN}[X]$. Expressions and their evaluation can be treated as a dynamic action structure $\mathsf{EXP}[X] = (M, X, A, \searrow)$, where

$$\begin{array}{rcl} M & = & (\mathsf{N}, +, 0) \\ X & = & \text{as before} \\ A_{m,n} & = & \text{Exp}_m^n \\ \searrow & & \text{defined below.} \end{array}$$

Now, for $e \in \text{Exp}_m$, write $e[\vec{d}]$ to mean the result of replacing the m formal variables $\vec{y}$ simultaneously in e by the m expressions $\vec{d}$; write $\{\vec{d}/\vec{x}\}e$ to mean the result of replacing the

k global variables $\vec{x}$ simultaneously in e by the k expressions $\vec{d}$. Each member of $A_{m,n}$ is an n-tuple of expressions, which we shall write as $\langle e_1, \ldots, e_n\rangle$. Then the operations in $\mathsf{EXP}[X]$ are

$$\begin{aligned}
\mathbf{id}_m &\stackrel{\text{def}}{=} \langle y_1, \ldots, y_m\rangle \\
\langle\vec{d}\rangle \cdot \langle\vec{e}\rangle &\stackrel{\text{def}}{=} \langle\vec{e}\,[\vec{d}\,]\rangle \\
\langle\vec{d}\rangle \otimes \langle\vec{e}\rangle &\stackrel{\text{def}}{=} \langle\vec{d},\, \vec{e}\,[y_{k+1}, \ldots, y_{k+m}]\rangle \\
&\qquad\qquad \text{where } \langle\vec{d}\rangle : k \to l,\ \langle\vec{e}\rangle : m \to n \\
\mathbf{ab}_x\langle\vec{e}\rangle &\stackrel{\text{def}}{=} \langle y_1\,,\, \{y_1/x\}(\vec{e}\,[y_2, \ldots, y_{m+1}])\rangle\ .
\end{aligned}$$

Only the abstractors here are perhaps unfamiliar; the effect of $\mathbf{ab}_x$ is to transform x into the first formal variable, incrementing the indices of the others by 1.

We now turn to the reaction relation of EXP, which represents evaluation. First, suppose that for each value $v \in D$ there is a nullary operator $\underline{v}$; we write $\underline{v}\langle\ \rangle$ as $\underline{v}$ and call it a *constant*. For $a, a' : m \to n$, define $a \searrow^1 a'$ to mean that a' can be gained from a by replacing a single subexpression $\underline{op}\langle\underline{v_1}, \ldots, \underline{v_k}\rangle$ in a by the constant $\underline{op(v_1, \ldots, v_k)}$. Then define reaction to be the transitive reflexive closure of $\searrow^1$ so that $\searrow \stackrel{\text{def}}{=} (\searrow^1)^*$.

Now of course we look for a natural homomorphism $\Phi : \mathsf{EXP}[X] \to \mathsf{FUN}[X]$ sending each expression to its denotation – a function. What is this denotation? For $e \in \mathrm{Exp}_0$ (an expression with no formal variables) it is given by $[\![e]\!]$, where

$$\begin{aligned}
[\![x]\!]\varrho &\stackrel{\text{def}}{=} \varrho x \\
[\![\underline{op}\langle e_1, \ldots, e_k\rangle]\!]\varrho &\stackrel{\text{def}}{=} op([\![e_1]\!]\varrho, \ldots, [\![e_k]\!]\varrho)\ .
\end{aligned}$$

Then define Φ to act upon arities and names as the identity, and upon actions by

$$\Phi(\langle e_1, \ldots, e_n\rangle)\varrho \stackrel{\text{def}}{=} \lambda\vec{v}\,.\,\langle[\![e_1[\underline{\vec{v}}]]\!]\varrho, \ldots, [\![e_n[\underline{\vec{v}}]]\!]\varrho\rangle\ .$$

It is quite routine to check the Φ is a homomorphism; in particular, that it preserves reaction, i.e.

$$a \searrow_{\mathsf{EXP}[X]} a' \ \text{ implies } \ \Phi a = \Phi a'$$

(recalling that $\searrow_{\mathsf{FUN}[X]}$ is the identity).

Action structures are not limited to such familiar things as functions and expressions, as our next few examples show. Moreover, even these familiar cases yield more interest when we come to *effect structures* in the next section.

3.4 Synchronous CCS (1) Synchronous CCS [18] is a process calculus in which processes $P, Q, \ldots$ are built from actions $\alpha, \beta, \ldots$ by various constructions, among which are *prefixing* $\alpha.P$ and *product* $P \times Q$. The dynamics of processes is given by *labelled transitions* of the form $P \xrightarrow{\alpha} P'$; these are defined by transition rules such as

$$\alpha.P \xrightarrow{\alpha} P \qquad\qquad \frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\beta} Q'}{P \times Q \xrightarrow{\alpha\times\beta} P' \times Q'}\,.$$

The actions of Synchronous CCS are just elements of the free Abelian group generated by a fixed set Π of (positive) *particles* $\pi, \pi', \ldots$. Thus an instance of the product rule is

$$\frac{P \xrightarrow{\pi_1\overline{\pi_2}\pi_3} P' \qquad Q \xrightarrow{\pi_2\overline{\pi_3}} Q'}{P \times Q \xrightarrow{\pi_1} P' \times Q'}$$

because $\pi_1\overline{\pi_2}\pi_3 \times \pi_2\overline{\pi_3} = \pi_1$ is a product of actions.

Note that Synchronous CCS represents the interaction between two particles by using the group inverse operation. In the context of action structures we prefer to treat this reaction separately from the static algebra of actions; indeed for the π-calculus we shall find this separation to be necessary.

We now define a simple action structure in which reaction represents for the first time the *interaction* between two actions; that is, unlike in previous examples, it is possible for $a \otimes b$ to react when neither a nor b can react independently.

Let $\overline{\Pi} \stackrel{\text{def}}{=} \{\overline{\pi} \mid \pi \in \Pi\}$, the negative particles. Then we define the action structure $\mathsf{SCCS} = (M, X, A, \searrow)$, where

$$\begin{aligned} M &= \text{a singleton} \\ X &= \emptyset \\ A &= \text{multisets over } \Pi \cup \overline{\Pi}\ ; \end{aligned}$$

further, $\searrow$ is the least preorder on A such that for all $a \in A$ and $\pi \in \Pi$

$$a \cup [\,\pi, \overline{\pi}\,] \searrow a\ .$$

(Here $\cup$ means union of multisets; we write multisets in square brackets.) Then the identities, composition and tensor product of the action structure are just

$$\begin{aligned} \mathbf{id} &\stackrel{\text{def}}{=} \emptyset \\ \cdot, \otimes &\stackrel{\text{def}}{=} \cup\ , \end{aligned}$$

and there are no abstractors because the name-set X is empty. One can easily show that the action structure SCCS is freely generated by the elementary actions $[\,\pi\,]$, $[\,\overline{\pi}\,]$ $(\pi \in \Pi)$ and by the single reaction rule $[\,\pi\,] \otimes [\,\overline{\pi}\,] \searrow \mathbf{id}_0$.

Of course, this action structure only explains the *actions* of synchronous CCS, not its processes or process combinators. But some of the latter can indeed be explained via action structures, as we now show.

3.5 Synchronous CCS (2) In the full calculus Synchronous CCS, processes P are subject to three operations which involve the binding of particle names. They are:

$$\begin{aligned} \text{Restriction:}\quad & P\backslash\pi \\ \text{Renaming:}\quad & P[\pi'/\pi] \\ \text{Parametrisation:}\quad & F(\pi) \stackrel{\text{def}}{=} P\ . \end{aligned}$$

Now we wish to represent all binding uniformly by the abstraction operator **ab** (which in Section 5 will be extended to operate upon processes as well as actions). Indeed, we shall find it appropriate to deal with restriction and renaming entirely at the level of actions. In the case of Synchronous CCS, this means that we now introduce a name-set X which is just the particle-set Π. So we shall now cease to use the symbols π, Π, and shall only reintroduce them in Part II for the particles of the π-calculus.

Accordingly, we now generalize SCCS to $\mathsf{SCCS}[X] = (M, X, A, \searrow)$, where

$$\begin{aligned} M &\stackrel{\text{def}}{=} (\mathsf{N}, +, 0) \\ X &= \text{a denumerable set (the particle-names)}\ , \end{aligned}$$

and the actions A and reaction relation $\searrow$ are defined as follows. Each action $a : m \to n$ takes the form

$$(\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle \qquad (\,|\vec{x}| = m,\ |\vec{z}| = n\,)\ ,$$

where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are names in X, and S –the *body*– is a multiset over $X \cup \overline{X}$. We say that the names $\vec{x}$, $\vec{y}$ and $\vec{z}$ are respectively *imported*, *restricted* and *exported*, the first two being binding operations over the whole action. We shall assume always that the names $\vec{x}, \vec{y}$ are all distinct, and we do not distinguish a from any variant gained by change of bound names or by permutation of restricted names. Finally, reaction is defined by $\searrow \stackrel{\text{def}}{=} (\searrow^1)^*$, where $a \searrow^1 a'$ means that a' can be derived from a by deleting a single pair of complementary particles from its body.

Let us look at several simple cases of actions. Note that any of the four parts of an action may be empty, and we may then omit that part. Thus:

- The elementary interactions $[\,x\,], [\,\overline{x}\,]$ would be written in full as $(\,)(\,)\,[\,x\,]\,\langle\,\rangle$ and $(\,)(\,)\,[\,\overline{x}\,]\,\langle\,\rangle$, and have arity $0 \to 0$. The reaction relation is defined by the single rule $[\,x\,] \otimes [\,\overline{x}\,] \searrow \mathbf{id}_0$.
- Restriction is the action $\boldsymbol{\nu} : 0 \to 1 = (\boldsymbol{\nu}y)\,\langle y\rangle$, whose full form is $(\,)\,(\boldsymbol{\nu}y)\,\emptyset\,\langle y\rangle$.
- The name x as a datum is the action $\langle x\rangle : 0 \to 1$, whose full form is $(\,)(\,)\,\emptyset\,\langle x\rangle$.

Note that the datum $\langle x\rangle$ is distinct from the elementary action $[\,x\,]$.

The action structure operations are as follows. Let

$$\begin{array}{rcll} a & \equiv & (\vec{u})\,(\boldsymbol{\nu}\vec{v})\,R\,\langle\vec{w}\rangle & \quad (\vec{u}, \vec{v} \text{ not occurring in } b) \\ b & \equiv & (\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle & \quad (\vec{x}, \vec{y} \text{ not occurring in } a). \end{array}$$

Then:

$$\begin{array}{rcll} \mathbf{id}_m & \stackrel{\text{def}}{=} & (\vec{x})\,\langle\vec{x}\rangle & \quad (\,|\vec{x}| = m) \\ a \cdot b & \stackrel{\text{def}}{=} & (\vec{u})\,(\boldsymbol{\nu}\vec{v}\vec{y})\,R \cup \{\vec{w}/\vec{x}\}S\,\langle\{\vec{w}/\vec{x}\}\vec{z}\rangle & \\ a \otimes b & \stackrel{\text{def}}{=} & (\vec{u}\vec{x})\,(\boldsymbol{\nu}\vec{v}\vec{y})\,R \cup S\,\langle\vec{w}\vec{z}\rangle & \\ \mathbf{ab}_x a & \stackrel{\text{def}}{=} & (x\vec{u})\,(\boldsymbol{\nu}\vec{v})\,R\,\langle x\vec{w}\rangle & \quad (x \text{ not in } \vec{u} \text{ or } \vec{v}). \end{array}$$

Thus we see that abstraction is a combination of import and export.

Recall that SCCS is freely generated by the actions $[\,\pi\,], [\,\overline{\pi}\,]$ $(\pi \in \Pi)$. Analogously, every action in $\mathsf{SCCS}[X]$ can be expressed (using the above operations) in terms of:

$$\begin{array}{rcll} [\,x\,], [\,\overline{x}\,] & : & 0 \to 0 & \text{– elementary interactions } \ (x \in X) \\ \langle x\rangle & : & 0 \to 1 & \text{– datum } (x \in X) \\ \boldsymbol{\nu} & : & 0 \to 1 & \text{– new name} \\ \boldsymbol{\omega} & : & 1 \to 0 & \text{– old name.} \end{array}$$

These actions have all been already defined except $\boldsymbol{\omega}$, whose full form is $(x)(\,)\,\emptyset\,\langle\,\rangle$, which discards a name. (Think of $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ as "new" and "old", or "create" and "destroy", or "gain" and "lose".) But $\mathsf{SCCS}[X]$ is not *freely* generated by these actions; certain identities hold among them, e.g. $\langle x\rangle \cdot \boldsymbol{\omega} = \mathbf{id}_0$. [3]

[3] In [21] a complete set of identities is given, so that $\mathsf{SCCS}[X]$ is indeed the action structure freely generated from the above actions divided by the equivalence induced by the identities. These identities characterize an important subclass of action structures called *action calculi*.

We do not claim that this way of presenting the actions of Synchronous CCS is directly useful in applied work. Its purpose is to show how Synchronous CCS fits into the action structure framework, and it prepares for the treatment of the π-calculus in Part II.

3.6 CCS There are two differences between pure CCS [19] and Synchronous CCS which affect the way we should treat them using action structures. The first is that in the product $P \times Q$ of Synchronous CCS the processes P and Q must act in synchronization, whereas either of them may remain idle in the parallel composition $P \mid Q$ of CCS . This is reflected in the first two transition rules for CCS parallel composition:

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'},$$

while interaction is permitted by the third rule:

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}.$$

This third rule also reflects the other way in which CCS differs from the synchronous version; that is, the atomic action τ of CCS represents the occurrence of a single interaction between processes, while in Synchronous CCS the unit action 1 represents the simultaneous occurrence of any number of such interactions. In a sense the latter is closer to reality, for one needs a central scheduler to implement the interleaving of single interactions in CCS. But CCS was designed as an applicable theory, rather than as a programming language; the expansion theorem of CCS is easy to use just because interactions are treated singly. Moreover in [18] it was demonstrated that at the semantic level of observation equivalence it makes no difference whether single interactions of CCS are treated as interleaved or as simultaneous.

If single interactions were allowed to be simultaneous, then the action structure for CCS could be just that for Synchronous CCS; the only difference would be that the concurrent composition of CCS is treated as a derived operation in terms of product and delay (see Section 5.1 below). But to treat single interactions as interleaved only requires two slight changes in the action structure $\mathsf{SCCS}[X]$. So we define $\mathsf{CCS}[X]$ as a slight variant of $\mathsf{SCCS}[X]$, as follows. First we adjoin the special particle τ to the particles $X \cup \overline{X}$, and in a typical action $a = (\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle$ we take S to be a multiset of this larger class of particles. Second, we define $a \searrow^1 a'$ iff a' can be derived from a by replacing a single pair of complementary particles in S by τ (instead of deleting them). In Section 7, we shall see how this adaptation allows us to restrict attention to actions containing at most two particles.

3.7 Ports The constructions $\mathsf{SCCS}[Z]$ and $\mathsf{CCS}[Z]$ involve the complementary actions $[\,z\,]$ and $[\,\overline{z}\,]$ ($z \in Z$), with the reaction rule

$$[\,z\,] \otimes [\,\overline{z}\,] \searrow \mathbf{id}_0 \text{ (or } [\,\tau\,]) \,;$$

since the actions have arity $0 \to 0$ this can also be written

$$[\,\overline{z}\,] \cdot [\,z\,] \searrow \mathbf{id}_0 \text{ (or } [\,\tau\,]) \,.$$

But this only models the pure form of CCS, where $[\,z\,]$ or $[\,\overline{z}\,]$ is simply an indivisible action. By contrast, in CCS with value-passing we think of z as a port through which data can flow.

How does this fit into the framework of action structures? Well, Example 3.2 above –the action structure $\mathsf{FUN}[X]$ for functions– is about operations upon data; we can think of a datum of arity m (an m-tuple of values) being fed into a function of arity $m \to n$. Now, how can we allow ourselves to import and export m-ary data through a port z say, as well as passing them into and out of functions? It is natural to extend $\mathsf{FUN}[X]$ by introducing a set Z (disjoint from X) of port-names, and for each port name $z \in Z$ and each appropriate arity m to introduce actions

$$\begin{aligned} \mathbf{out}_z &: m \to 0 \\ \mathbf{in}_z &: 0 \to m \ . \end{aligned}$$

Now the actions $[\,\overline{z}\,]$ and $[\,z\,]$ of $\mathsf{SCCS}[Z]$ and $\mathsf{CCS}[Z]$ can be regarded as special cases of the above for which $m = 0$, since no data flows in those calculi. Therefore, generalising their reaction rule also, we introduce the reaction rule

$$\mathbf{out}_z \cdot \mathbf{in}_z \searrow \mathbf{id}_m \ .$$

This can also be written

$$\mathbf{in}_z \otimes \mathbf{out}_z \searrow \mathbf{id}_m \ ,$$

for it is a simple exercise to show that if $a : m \to 0$ and $b : 0 \to n$ in a strict monoidal category, then $a \cdot b = a \otimes b = b \otimes a$. This rule exactly expresses synchronized communication at a port.

Thus we are indicating how to superpose the action structure $\mathsf{SCCS}[Z]$ or $\mathsf{CCS}[Z]$ upon an arbitrary action structure, extending the latter by adding ports through which its "data" may pass. [4] Let us describe this superposition a little more formally. Let $B = (M, X, B, \searrow)$ be any action structure, and let Z (disjoint from X) be the ports we wish to add. To construct $B^{(Z)} = (M', X', B', \searrow')$, the action structure B freely extended by addition of the ports Z, we perform the following steps.

First define

$$\begin{aligned} M' &= M \times \mathsf{N} && \text{– Cartesian product of monoids} \\ X' &= X \cup Z && \text{– a disjoint union.} \end{aligned}$$

Let $\Phi : X \to X'$ and $\Phi : M \to M'$ be the injections given by $\Phi x = x$ and $\Phi m = (m, 0)$. The arities in X' are $x : (m, 0)$ for each $x : m \in X$, and $z : (\epsilon, 1)$ for each $z \in Z$.

Next, the actions B' are as follows:

1. There is an injection $\Phi : B \to B'$; if $a : m \to n$ in B then $\Phi a : \Phi m \to \Phi n$ in B'.

2. There are actions corresponding to those of $\mathsf{SCCS}[Z]$:

$$\begin{aligned} \mathbf{in}_z &: (\epsilon, 0) \to (m, 0) && \text{– import data through port } z \in Z \\ \mathbf{out}_z &: (m, 0) \to (\epsilon, 0) && \text{– export data through port } z \in Z \\ \langle z \rangle &: (\epsilon, 0) \to (\epsilon, 1) && \text{– the port } z \in Z \text{ as a datum} \\ \boldsymbol{\nu} &: (\epsilon, 0) \to (\epsilon, 1) && \text{– created port} \\ \boldsymbol{\omega} &: (\epsilon, 1) \to (\epsilon, 0) && \text{– discarded port.} \end{aligned}$$

[4] The quotation marks are to remind us that some action structures may be abstract and not concerned with data; nonetheless our construction makes sense formally even in these cases.

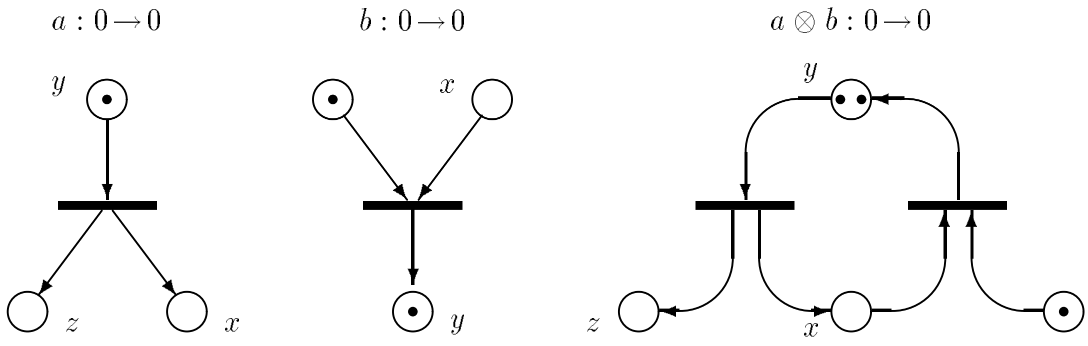


Figure 1: Two PT-nets and their product

Finally, the reaction relation is generated by the rule

$$\Phi a \searrow' \Phi b \text{ iff } a \searrow b$$

ensuring that B is a sub-actionstructure of $B^{(Z)}$, together with the rule

$$\mathbf{out}_z \cdot \mathbf{in}_z \searrow' \mathbf{id}_{(m,0)} \qquad (z \in Z, m \in M).$$

This construction remains to be detailed more precisely. Note in particular that we have *not* just taken the Cartesian product of B with $\mathsf{SCCS}[Z]$; for the port actions $\mathbf{in}_z$ and $\mathbf{out}_z$ are indexed by $z \in Z$, but their arities are drawn from B.

Note also that although we allow a port z to appear as a datum $\langle z \rangle$, it may not be exported through another port $w \in Z$; for the arities $\langle z \rangle : (\epsilon, 0) \to (\epsilon, 1)$ and $\mathbf{out}_w : (m, 0) \to (\epsilon, 0)$ forbid the composition $\langle z \rangle \cdot \mathbf{out}_w$. This is exactly the limitation of CCS with value-passing – namely, that ports cannot be passed as messages. (Indeed we expect to be able to present an action structure for CCS with value-passing just by extending $\mathsf{FUN}[X]$, for suitable data D, by adding ports in this way.) The π-calculus, for which we define action structures in Part II, corresponds exactly to the removal of this limitation.

3.8 Petri nets The monoidal structure of Petri nets was first demonstrated by Meseguer and Montanari [17]. They model each place-transition net (PT-net) as a monoidal category, whose objects are markings and whose morphisms correspond to firing sequences. It is natural to ask whether the extra ingredients in an action structure can play a rôle in the formulation of Meseguer and Montanari. But they already model the dynamics of a net N by the morphisms of the monoidal category corresponding to N, so there is no rôle for the dynamics of action structures. Nor is there an obvious rôle for abstractors at the level of a single net N.

But there is a natural action structure at a higher level, where each marked PT-net N corresponds to an action. Here the abstractors find a good rôle to play, while an atomic reaction $\searrow^1$ corresponds to the firing of a single transition; if $a \searrow^1 a'$ then a' is the same net as a with a different marking.

This action structure is treated in more detail in [21]; here we sketch it, for comparison with the previous examples. First we take $(\mathsf{N}, +, 0)$ to be the arity monoid, and take a name-set X infinite at each arity. Then an action of arity $0 \to 0$ is just a marked PT-net some of whose places bear a name from X, no two places bearing the same name. Call this a *place-named* net. Figure 1 shows two place-named nets and their product; the product is formed just by coalescing like-named places.

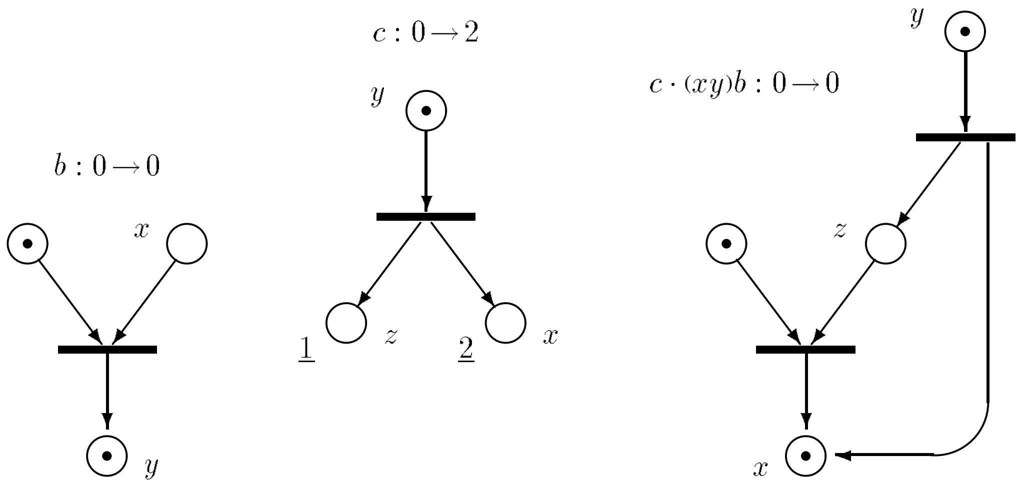


Figure 2: The composition of PT-net actions

We now turn to actions of arbitrary arity. An action $d : m \to n$ consists of a place-named net N together with a vector $\vec{x}$ of m distinct names representing "bound" or parametric places in the net, and a map (not necessarily injective) from $\{1, \ldots, n\}$ into the places of N indicating the "target" places. Typically we would write $d = (\vec{x})c$ with $c : 0 \to n$. (We allow alpha-conversion of bound names in the usual way.) Figure 2 shows $c : 0 \to 2$ formed from a by targetting two places.

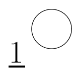
To form the product of two actions $a : 0 \to m$ and $b : 0 \to n$ with zero source arity, simply increment the target labels in b by m and proceed to coalesce like-named places as in Figure 1. To form the product of two arbitrary actions $(\vec{x})a$ and $(\vec{y})b$ first ensure by alpha-conversion that the bound names in one action do not occur free or bound in the other, then form $a \otimes b$ as above; the result is $(\vec{x}\vec{y})(a \otimes b)$.

To compose two actions, e.g. to form $c \cdot (xy)b$ where b and c are as in Figure 2, first alpha-convert to disjoin bound names as for product; then coalesce each target place in the first action with the corresponding bound place in the second action, removing the target labels of the first action and the bound names of the second.

To form the abstraction $\mathbf{ab}_x d : 1{+}m \to 1{+}n$ for arbitrary $d : m \to n$, first increment the target labels in d by one; then target the place named x as 1 (adding an unconnected place x if there is none) and bind x.

Finally, the reaction relation is generated by $\searrow^1$ which represents the firing of a single transition. With a as in Figure 1, we have $a \searrow^1 a'$ where both places x, z are marked in a', and a' is inactive. Moreover it is easily seen that $a \otimes b \searrow^1 a' \otimes b$, but the latter is *not* inactive.

So our action structure contains not only place-named nets, but also nets whose places are targeted and/or bound. Because of this, many operations on nets can be represented algebraically. A simple example is hiding a place, i.e. removing its name and thus fixing its pre- and post-transitions (since it may no longer be coalesced with a like-named place). This is just our familiar restriction operator $\boldsymbol{\nu}$. In fact $\boldsymbol{\nu} : 0 \to 1$ is the action



consisting of a single place – unnamed, unmarked and targeted; to hide the place z in any $c : 0 \to n$ we just form $\boldsymbol{\nu} \cdot (z)c : 0 \to n$. Other simple operations are: increasing the marking at a named place; renaming a place; coalescing two differently-named places. Also all place-named nets can be formed by product alone, from simple nets with at most one transition.

A dual action structure to the above allows the naming of transitions, instead of places. These two action structures can also be conflated, allowing naming (hence coalescing) both of transitions and of places.

Much work has been done on algebras of nets. Two good examples are the algebra AFP of Cherkasova and Kotov [9], and the box calculus of Best, Devillers and Hall [7] which is oriented towards defining concurrent programming languages. The action structure approach must be placed in that context, but this task is not addressed here; we have only been concerned to show that action structures yield at least one way of treating Petri nets algebraically. Its use of parametrisation appears to be new.

The relationship must also be examined between the present notion of action structure homomorphism, specialised to Petri nets, and the categorical treatment of nets and their morphisms by Winskel [28] and others.

4 Effects and effect structures

A central aim of action structures is to make explicit the way in which an interaction exerts an effect upon its participants. Recall from Section 1 our example in the π-calculus; the transition

$$\big((\boldsymbol{\nu}x)\,\overline{p}x.P\big) \mid p(y).Q \;\longrightarrow\; (\boldsymbol{\nu}x)\big(P \mid \{x/y\}Q\big)$$

is represented roughly speaking by the reaction

$$\big((\boldsymbol{\nu}x)\cdot\overline{p}x\big)\otimes p(y) \searrow (\boldsymbol{\nu}x)\cdot\big(\mathbf{id}\otimes\{x/y\}\big)$$

in an appropriate action structure.

We shall refer to degenerate actions, such as the result of this reaction, as *effects*; our aim is to see how effects are related to other actions. In passing, note that one might expect the result of a reaction of a product to be itself a product, whose factors are the separate effects exerted by the reaction on each of the participants P and Q. But that is not the case here. A central idea of the π-calculus is that the effect of a communication may be a new liaison between a pair of agents, i.e. a restriction; this is an effect upon the *pair*, not upon its individual members.

What distinguishes effects from other actions? *Inertia*, the inability to contribute to further reaction, is one distinguishing property.

4.1 Definition (inactivity, inertia) An action a is *inactive* if $a \searrow a' \Rightarrow a' = a$. An action a is *inert* if, whenever $b\cdot a \searrow c$, then there exists b' such that $b \searrow b'$ and $c = b'\cdot a$. ■

Thus if a is inert then any reaction of $b\cdot a$ must be due to b alone.

4.2 Proposition If an action is inert then it is inactive.

Proof The identity **id** is inactive by assumption; so take $b = \mathbf{id}$ in the definition. ■

4.3 Proposition Each unit action **id** is inert. If a and b are inert then so is $a\cdot b$. ■

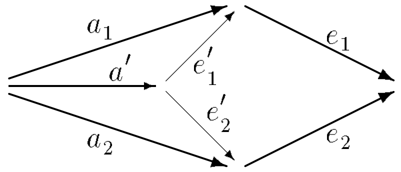


Figure 3: Decomposition for a postcomponent

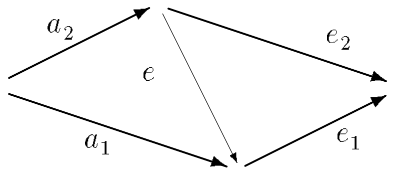


Figure 4: Preordered decompositions: $(a_1, e_1) \leq (a_2, e_2)$

Inertia is *not* symmetric with respect to composition. If a is inert then $a \cdot b$ may still have reactions not arising from b alone. For example a may be roughly speaking a substitution, which will indeed be an inert action, and the *effect* of this upon b may induce a reaction.

As well as demanding that each effect is an inert action, we also demand certain properties of the *set* of effects in an action structure. This will not fully determine the effects; indeed it will be useful to consider different subsets of actions as effects.

4.4 Definition (postcomponent) Let A be an action structure, and E a sub-actionstructure of A. Then E is a *postcomponent* of A if whenever $a = a_1 \cdot e_1 = a_2 \cdot e_2$, where $a_i \in A$ and $e_i \in E$ ($i = 1, 2$), then there exist $a' \in A$ and $e'_1, e'_2 \in E$ such that (see Figure 3):

$$a_i = a' \cdot e'_i \ \ (i = 1, 2) \ \text{ and } \ e'_1 \cdot e_1 \ = e'_2 \cdot e_2.$$

Further, if all members of E are inert then E is an *effect structure* for A. ■

We continue to use a, b, c, d to range over all actions; when the effect structure E is understood we use e, f, g to range over effects.

Note that "postcomponent" is a purely static notion. The condition that E is a postcomponent is about decompositions $a = a' \cdot e$ of an action, where $e \in E$. A few more definitions will clarify the rôle played by decompositions.

4.5 Definition (decomposition) Let E be a sub-actionstructure of the action structure A. Then the pair (a', e) is a *decomposition of a for E* if $a = a' \cdot e$ and $e \in E$. We define the following preorder over decompositions for E:

$$(a_1, e_1) \leq (a_2, e_2) \ \text{ if } \ a_1 = a_2 \cdot e \ \text{ and } \ e \cdot e_1 = e_2 \ \text{ for some } \ e \in E.$$

(See Figure 4.) We say that a decomposition (a^*, e^*) of a for E is *maximal* if $(a', e) \leq (a^*, e^*)$ for any other decomposition (a', e) of a for E. ■

Thus the postcomponent condition requires that the decompositions of any action form a directed set under $\leq$. In establishing that a given sub-actionstructure of A is an effect structure, this condition is usually the hardest to check. It is helpful to look for *maximal* decompositions, since they often exist for interesting postcomponents. Their importance is as follows:

4.6 Proposition If E is sub-actionstructure of A, and every $a \in A$ has a maximal decomposition for E, then E is a postcomponent of A. ■

In fact something even stronger often holds for postcomponents which we wish to use as effect structures. In general, for given a, the decomposition (a, e) may be maximal for some effects e but not for others. But for many effect structures there is a subset of A which behaves better than this:

4.7 Definition (pure action) Let E be a sub-actionstructure of the action structure A. Then a is *pure for* E if for every $e \in E$ the decomposition (a, e) is maximal. We say that the decomposition (a^*, e^*) is *pure for* E if a^* is pure for E and $e^* \in E$. ■

In verifying that some given sub-actionstructure E of inert actions is an effect structure it is a good strategy first to identify the pure elements, and then to show that every action a has a pure decomposition, which is of course maximal. The strategy may not always work –there may not be enough pure elements– but it works in two of the cases we examine in Section 4.8 below. We shall also find in Section 6 that the concept of bisimilarity between processes is simpler when the the underlying action structure always has pure decompositions.

Let us denote by P the set of elements pure for E in A. Having identified P as an interesting set of actions, should we expect it to be a sub-actionstructure of A? It turns out to be so in the two examples just mentioned, where pure decompositions always exist. Under one or two extra conditions, which also hold in those examples, the pair (P, E) of sub-actionstructures becomes a *factorisation* of A, in the sense of Freyd and Kelly [10]. I am grateful to John Power for prompting me to look in this direction. There is a duality between the two members of a factorisation; we therefore have –at least in these examples– a duality between pure actions and effects, which may shed extra light on these concepts.

4.8 Examples What possible effect structures exist for action structures in general, and in particular for the examples in Section 3? Certainly the units $E = \{\mathbf{id}_m \mid m \in M\}$ always form an effect structure. It is not a very interesting one, since every $a : m \to n$ possesses only one effect in E, namely $\mathbf{id}_n$. Indeed, there is advantage in finding a larger effect structure; for we shall see later that this makes bisimilarity a more generous congruence relation over processes.

We turn now to the example action structures of Section 3. Both FUN and $\mathsf{FUN}[X]$ are static, i.e. reaction is the identity, so all actions are inert; therefore these action structures are effect structures for themselves. We would need to know more specific properties of the functions to determine any other effect structure here, apart from the units.

The evaluation of expressions, $\mathsf{EXP}[X]$ in Section 3.3, is immediately more interesting. First, there are many inactive actions which are not inert; $\langle y_1, y_2 \underline{+} \underline{4}\rangle$ is inactive, but if we precompose $\langle \underline{2}, \underline{3}\rangle$ to it we get $\langle \underline{2}, \underline{3} \underline{+} \underline{4}\rangle$ – which is active. However, any action $\langle e_1, \dots, e_n\rangle$ containing only local variables and constants (i.e. no global variables or non-nullary operators) turns out to be inert, provided it contains each local variable at most once. An example is $\langle y_2, y_1, \underline{4}\rangle$. To see why each y_i should appear at most once, consider $a = \langle y_1, y_1\rangle$; if $b = \langle \underline{2} \underline{+} \underline{3}\rangle$, we have

$$b \cdot a = \langle \underline{2} \underline{+} \underline{3}, \underline{2} \underline{+} \underline{3}\rangle \searrow \langle \underline{5}, \underline{2} \underline{+} \underline{3}\rangle$$

and we cannot find b' for which $b \searrow b'$ and $\langle \underline{5}, \underline{2} \underline{+} \underline{3}\rangle = b' \cdot a$. [5]

[5] Of course if we take $b' = \langle \underline{5}\rangle$ then we *do* have the weaker property $\langle \underline{5}, \underline{2} \underline{+} \underline{3}\rangle \searrow b' \cdot a$. This suggests that a

So do the actions containing only local variables and constants, and each y_i at most once, form an effect structure? They do indeed; but it turns out to be one in which maximal decompositions do not always exist. The reader may enjoy checking these facts, and also verifying that effects cannot contain global variables.

Now consider the subset of these actions containing each y_i *exactly* once. It is easy to see that they form a sub-actionstructure. And in contrast to the previous case we find that pure decompositions always exist, so we have an effect structure by Proposition 4.6. The pure actions are of the form $\langle d_1, \ldots, d_m \rangle$ where no d_j is a constant $\underline{v}$. For a pure decomposition (a^*, e^*) of any $a = \langle e_1, \ldots, e_n \rangle$, take a^* to be some permutation $\langle e_{i_1}, \ldots, e_{i_m} \rangle$ of the expressions e_i excluding any which is a constant.

For SCCS(Section 3.4) it's easy to see that the only effect structure is $\{\mathbf{id}_0\}$; for a non-empty multiset cannot be inert. But $\mathsf{SCCS}[X]$ and $\mathsf{CCS}[X]$ (Sections 3.5 and 3.6) are more interesting. Let E contain all actions $(\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle$ in which the multiset S is empty; then all members of E are certainly inert. Also E is indeed an effect structure. Again, this can be shown by identifying the pure actions; they turn out to be all those $(\vec{x}))\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle$ in which every bound name (one of $\vec{x}$ or $\vec{y}$) occurs in $\vec{z}$, and every restricted name (one of $\vec{y}$) also occurs in S.

These last two examples of effect structures are strikingly different; so it appears that the notions of effect and effect structure are quite flexible.

4.9 Inertia reconsidered According to Definition 4.1, for a to be inert means that a cannot create reactions if you *postcompose* it, i.e. if you form $c \cdot a$ for some c. But our interest in effects is based on the idea that an effect e can create reactions if you *precompose* it, i.e. if you form $e \cdot c$ for suitable c. Now it turns out in all our examples that an effect e can create reactions *only* if you precompose it, e.g. not if you juxtapose it as in $e \otimes c$. So why do we not take this stronger condition to be the definition of inertia? Does it make much difference if we do?

The answer is: There may be fewer inert actions with this stronger condition, but it does not affect the concept of effect structure at all. Let us make this claim precise.

4.10 Definition (preaction) A *preaction* φ is a unary operation on actions built from operations of the form $c \cdot (-)$, $c \otimes (-)$, $(-) \otimes c$ and $\mathbf{ab}_x(-)$ where c is any action. ■

The term "preaction" is chosen to indicate that *precomposition* $c \cdot (-)$, but not *postcomposition* $(-) \cdot c$, is allowed in forming preactions. Structural congruence and the reaction relation extend in an obvious way to preactions. We now define strong inertia using preactions, where in definingo inertia we used only precomposition. So strong inertia clearly implies inertia.

4.11 Definition (strong inertia) An action a is *strongly inert* if whenever $\varphi(a) \searrow b$, then $b = \varphi'(a)$ for some preaction φ' such that $\varphi \searrow \varphi'$. ■

Now we show that it would make no difference to the notion of effect structure, Definition 4.4, if we demanded that its members be *strongly* inert.

form of inertia weaker than Definition 4.1 might be useful, with the condition $c = b' \cdot a$ weakened to $c \searrow b' \cdot a$. It remains to be seen how much of the ensuing theory goes through with this weaker notion. But it is worth noting that for the other examples discussed in Section 3 the weakening does not seem to increase the set of inert actions.

4.12 Lemma Every preaction φ can be expressed as $\varphi = a \cdot \psi$, where ψ is a preaction formed by the operations **id** $\otimes (-)$, $(-) \otimes$ **id** and $\mathbf{ab}_x(-)$.

Proof By induction on the structure of preactions. To take just the most interesting case, consider the preaction $c \otimes \varphi$. By assumption $\varphi = a \cdot \psi$ where ψ is of the required form, so

$$\begin{aligned} c \otimes \varphi &= c \otimes (a \cdot \psi) \\ &= (c \otimes a) \cdot (\mathbf{id} \otimes \psi) \end{aligned}$$

which is of the required form. ∎

4.13 Proposition Let E be an effect structure. Then all its members are strongly inert.

Proof Let $e \in E$ and let φ be a preaction such that $\varphi(e) \searrow b$. Now let $\varphi = a \cdot \psi$ as in the lemma; so $a \cdot \psi(e) \searrow b$. But $\psi(e) \in E$ is inert, so for some a' we have $a \searrow a'$ and $b = a' \cdot \psi(e)$. But $a' \cdot \psi$ is a preaction and $\varphi \searrow a' \cdot \psi$. So we have shown e to be strongly inert. ∎

Thus we have lost nothing in demanding only the weaker condition of inertia in defining an effect structure; moreover this weaker condition is simpler to verify in examples.

Preactions will turn up again in Section 7, where we define the notion of *incident*.

5 Process structure: statics

We can use action structures in more than one way to model concurrent processes. In the first way, each *single transition* of a process would be modelled by an action. At the higher level of the processes, these single transitions may be considered atomic, even though the detailed performance of an action –i.e. the reaction within it– may be highly complex.

A second approach is to model a *whole process* by an action, as well as its individual transitions. This approach depends upon finding ways of building a higher-level action structure (for the processes) from a lower-level one (for the transitions). *Action calculi* [21], a subclass of action structures, make considerable progress in this direction.

Here we take the first approach. We show how certain control operations can be added to an arbitrary action structure, to yield a process calculus. This yields a generic process calculus, and we can carry its theory forward a fair way independently of the underlying action structure.

Thus we bring some unity to the study of process calculi. Specifically, we can unite the study of CCS, Synchronous CCS and the π-calculus. Other calculi such as ACP [3] and CSP [14] may also fit into the framework; this remains to be seen.

The two approaches do not conflict. In fact, the uniform treatment in this paper of bisimilarity congruence in process calculi is a preparation for studying bisimilarity in action calculi. For in contrast with actions which are arrows $a : m \to n$ with both a source and a target arity, the processes $P : m$ which we are about to introduce have only a source arity; in fact they correspond to actions $a : m \to \epsilon$ with trivial target arity. One reason for adopting this constraint is that we remain closer to existing process calculi; another is that the technical problems of bisimilarity are easier to handle.

5.1 Process syntax We study processes over an arbitrary action structure, whose actions will be denoted by $a, b, \ldots$ as before. Processes, denoted by $P, Q, \ldots$, are given by the syntax

$$\begin{aligned} P \;\; ::= \;\; & a \cdot P \;\big|\; P \otimes P \;\big|\; (x)P \\ & \big|\; \bullet P \;\big|\; \mathbf{0} \;\big|\; P + P \;\big|\; \boldsymbol{\partial} P \;\big|\; !P \;. \end{aligned}$$

The first three constructions lift the operations of an action structure to process level, but allowing composition only between an action and a process rather than the composition $P \cdot Q$ of two processes. [6] Note that we use $(x)P$ rather than $\mathbf{ab}_x P$ for abstraction over processes.

The fourth construction $\bullet P$ represents what we shall call *committal*. Once we have defined reaction for processes, then we shall find that a process P may be capable of many different reactions of the form

$$P \searrow a \cdot \bullet P'$$

which we shall call *commitments*. To establish a link with CCS syntax we shall elide the "$\cdot$" before "$\bullet$", writing

$$P \searrow a \bullet P'.$$

Each commitment represents P's ability to perform a single action a, considered atomic at process level, with continuation P'. Moreover, if (a', e) is a decomposition of a then our theory will reflect the intuition that the effect e exerts an influence upon the continuation P'. The need to write down this influence explicitly, as $e \cdot P'$, is one reason for admitting the composition construction at process level.

The last five constructions from committal ($\bullet$) onwards represent control, and we call them *control terms*; equally we call $\bullet$, $\mathbf{0}$, $+$, $\boldsymbol{\partial}$ and ! *control operators*, and refer to them respectively as committal, zero, sum, delay and replication. The last four are drawn from Synchronous CCS and the π-calculus; but they do not depend on the nature of the *actions* in those calculi, so it makes sense to use them more generally.

The set of five control operators is not claimed to be complete or fundamental; indeed the notion of action structure has greater claim to this virtue. But they work well in their parent calculi, so they serve as exemplars for adding control structure uniformly to action structure. Recall from [18] that, in the framework of Synchronous CCS, the parallel composition operator "|" of CCS can be recovered by the definition

$$P \mid Q \stackrel{\text{def}}{=} P \times \boldsymbol{\partial} Q + \boldsymbol{\partial} P \times Q\ .$$

The dynamics of each operator is presented in Section 6; the remainder of this section is devoted to statics.

5.2 Arity Processes do not constitute an action structure; in this approach they are not actions. Instead, each process has an arity $m \in M$ given by the rules

$$\frac{a : m \to n \qquad P : n}{a \cdot P : m} \qquad \frac{P : m \qquad Q : n}{P \otimes Q : m \otimes n} \qquad \frac{x : k \qquad P : m}{(x)P : k \otimes m}$$

$$\frac{P : m}{\bullet P : m} \qquad \mathbf{0} : m \qquad \frac{P : m \qquad Q : m}{P + Q : m} \qquad \frac{P : m}{\boldsymbol{\partial} P : m} \qquad \frac{P : \epsilon}{!P : \epsilon}$$

These are mostly straightforward. A process P with non-ϵ arity should be thought of as parametric; its parameter would typically be provided by an effect precomposed to P. Note that $!P$ is only allowed when $P : \epsilon$. Why? Well, we expect $!P$ to be behaviourally equivalent to $(P \otimes\ !P) + \bullet\ !P$, so the arities of these two must be the same. Bearing in mind the rules for sum and committal, this implies that the two arities $P : k$ and $!P : m$ satisfy the equations $m = k \otimes m = k$, which implies $m = k = \epsilon$ (at least if the arity monoid is free).

[6] As remarked earlier, we do not here consider processes as arrows with both source and target arity; thus we avoid composition of processes. This makes the present theory simpler; in particular, it admits the simple head normal form of Definition 5.6.

5.3 Structural congruence We impose a structural congruence, $\equiv$, upon the syntax of processes. It is defined by the following laws, i.e. it is the smallest congruence satisfying them:

$$\begin{array}{rcl} P & \equiv & \mathbf{id} \cdot P \\ a \cdot (b \cdot P) & \equiv & (a \cdot b) \cdot P \\ (a \cdot P) \otimes (b \cdot Q) & \equiv & (a \otimes b) \cdot (P \otimes Q) \\ (x)(a \cdot P) & \equiv & (\mathbf{ab}_x a) \cdot (x)P \,. \end{array}$$

As far as possible these are the laws of action structures lifted to process level. In particular they allow us to confuse the process-level and action-level operations with impunity.

There are no equational laws for the control operators. We could have imposed some, e.g. $P + Q \equiv Q + P$, $\ldots$, but we prefer to attain these equations for a *behavioural* (rather than structural) congruence. Thus the meaning of the control operators is entirely provided by the process dynamics in Section 6.

We shall need the following notion: An occurrence in P of an action a or process term Q is called a *ready* occurrence if it is not within any control term. For example a occurs readily in $P \otimes a \bullet Q$ but not in $P + a \bullet Q$.

Henceforth we shall usually omit the composition symbol, writing $a \cdot b$ as ab and $a \cdot P$ as aP. We give (elided) composition highest binding power, and all other binary operators lowest binding power; so $ab \otimes cd$ means $(ab) \otimes (cd)$, $\mathbf{ab}_x cd$ means $\mathbf{ab}_x(cd)$, $(x)P \otimes Q$ means $((x)P) \otimes Q$ and $\boldsymbol{\partial} P + Q$ means $(\boldsymbol{\partial} P) + Q$.

5.4 Passive processes The dynamics of processes will be a cooperation between reactions due to the underlying action structure, and control reductions. The former can only be contributed by ready occurrences of actions a. Intuitively we can say that a process is passive (cannot react) if no $a \neq \mathbf{id}$ has a ready occurrence. Thus if $a \neq \mathbf{id}$ we would say that $a \bullet P \otimes \ \bullet Q$ is active; on the other hand $a \bullet P + \ \bullet Q$ is passive, because a control reduction is needed to allow a to react. Formally:

5.5 Definition (passive process) The *passive* processes are the smallest set closed under structural congruence such that

- All control terms are passive;
- If P, Q are passive, then so are $P \otimes Q$ and $(x)P$. ■

(The important feature here is the omission of composition, aP, from the second clause.) It will follow from Lemma 6.6 that no reaction can occur in an passive process, until a control reduction has occurred.

We shall use $H, J, K, \ldots$ to range over passive processes. The next few results show that any P can be uniquely expressed in the form aH, up to structural congruence.

5.6 Definition (head normal form) If $P \equiv aH$, where H is passive, then the latter is called a *head normal form* (*hnf*) of P. ■

5.7 Proposition Every process possesses a head normal form.

Proof By induction on the structure of process terms. ■

This justifies our structural congruence laws (Section 5.3), since they seem to be the minimum which will yield such a result. The next proposition strengthens this claim:

5.8 Proposition Head normal forms are unique up to structural congruence; that is, if $aH \equiv bJ$ then $a = b$ and $H \equiv J$.

Proof We first show that $aH \equiv bJ$ implies $a = b$. For this purpose, we use an interpretation $\mathcal{I}$ of process terms in the underlying action structure; $\mathcal{I}$ is defined in the obvious way for composition, product and abstraction over processes, and takes all control terms to the unit action. Thus

$$\begin{aligned} \mathcal{I}(aP) &\stackrel{\text{def}}{=} a\mathcal{I}(P) \\ \mathcal{I}(P \otimes Q) &\stackrel{\text{def}}{=} \mathcal{I}(P) \otimes \mathcal{I}(Q) \\ \mathcal{I}((x)P) &\stackrel{\text{def}}{=} \mathbf{ab}_x \mathcal{I}(P) \\ \mathcal{I}(R) &\stackrel{\text{def}}{=} \mathbf{id}_m \qquad \text{if } R \text{ is a control term.} \end{aligned}$$

(In the last case m is the arity of the control term.)

Note that if $P : m$ then $\mathcal{I}(P) : m \to n$ for some n which is determined by the translation. It is easy to check that the four laws of structural congruence preserve $\mathcal{I}$, i.e. $P \equiv Q$ implies $\mathcal{I}(P) = \mathcal{I}(Q)$. From Definition 5.5 it follows that $\mathcal{I}(H) = \mathbf{id}$ for any passive process H. Hence from $aH \equiv bJ$ we infer

$$a = \mathcal{I}(aH) = \mathcal{I}(bJ) = b.$$

Next we show that $aH \equiv bJ$ implies $H \equiv J$. For any P, let $\widehat{P}$ be the result of replacing all ready occurrences of actions by the unit action. For example if P is $R \otimes a\boldsymbol{\partial}(bQ)$ then $\widehat{P}$ is $\widehat{R} \otimes \boldsymbol{\partial}(bQ)$. In general the arity of $\widehat{P}$ will differ from that of P, but not in the case of an passive term H – in fact, $\widehat{H} \equiv H$.

Now suppose that $P \equiv Q$ is proved by a single application of one of the four laws of structural congruence to a subterm of P. Then it can be checked that also $\widehat{P} \equiv \widehat{Q}$. Essentially this is because in each law a ready occurrence on one side corresponds uniquely to a ready occurrence on the other side. So in general $P \equiv Q$ implies $\widehat{P} \equiv \widehat{Q}$. Thus from $aH \equiv bJ$ we infer, as required,

$$H \equiv \widehat{aH} \equiv \widehat{bJ} \equiv J\ .$$ ■

Finally, we shall need the following simple proposition about committal:

5.9 Proposition If $\bullet P \equiv \bullet Q$ then $P \equiv Q$.

Proof For any R, let $\widehat{R}$ be the result of simultaneously replacing every ready occurrence of a commitment $\bullet S$ in R by S itself. By means similar to the second half of the proof of Proposition 5.8, we see that $R_1 \equiv R_2$ implies $\widehat{R_1} \equiv \widehat{R_2}$. But $\widehat{\bullet P}$ is P, so the result follows. ■

6 Process structure: dynamics

Our presentation of process dynamics is best introduced by contrast with what was done for CCS and Synchronous CCS. Consider the following transition of Synchronous CCS:

$$(\pi.P + \pi'.Q) \times \overline{\pi}.R \xrightarrow{1} P \times R\ ;$$

according to [18] it is derived by three transition rules, whose general form is:

$$\alpha.P \xrightarrow{\alpha} P \qquad\qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\beta} Q'}{P \times Q \xrightarrow{\alpha\times\beta} P' \times Q'}.$$

Here we wish to derive such transitions in another way; we wish to distinguish the contributions made to any transition by *control* from those made by *action*. We shall use the relations $\underset{c}{\searrow}$ and $\underset{a}{\searrow}$ for these respective contributions. So taking SCCS as the underlying action structure, the above transition will be derived in three stages using the rules given below in 6.2 and 6.3:

$$\begin{array}{rcll}
(\pi \bullet P + \pi' \bullet Q) \otimes \overline{\pi} \bullet R & \underset{c}{\searrow} & \pi \bullet P \otimes \overline{\pi} \bullet R & \text{by a rule for } + \\
& \equiv & (\pi \otimes \overline{\pi})(\,\bullet P \otimes\ \bullet R) & \\
& \underset{c}{\searrow} & (\pi \otimes \overline{\pi}) \bullet (P \otimes R) & \text{by a rule for } \bullet \\
& \underset{a}{\searrow} & \mathbf{id} \bullet (P \otimes R) & \text{by the rule for } \underset{a}{\searrow}.
\end{array}$$

where we have written π for the action $[\,\pi\,]$ etc. Thus a single process-transition is represented by a sequence of reductions. We now proceed to the formal definitions.

6.1 Definition (context, ready context) A *context* $\mathcal{C}$ is a process term with a single hole, denoted by [ ], to be filled by a process term. We write $\mathcal{C}[P]$ for result of placing P in the context $\mathcal{C}$. If the hole occurs readily then $\mathcal{C}$ is a *ready* context. ■

$[\ ] \otimes Q$ is an example of a ready context; $[\ ] + Q$ is not ready. Structural congruence extends to contexts, and preserves readiness.

6.2 Definition (action preorder) The *action preorder* $\underset{a}{\searrow}$ over processes is the least preorder closed under structural congruence, such that for all ready $\mathcal{C}$

$$a \searrow a' \ \text{ implies } \ \mathcal{C}[aP] \underset{a}{\searrow} \mathcal{C}[a'P].$$

We use $\overset{1}{\underset{a}{\searrow}}$ to represent a single use of this rule, so that $\underset{a}{\searrow}\ =\ (\overset{1}{\underset{a}{\searrow}})^*$. ■

6.3 Definition (control preorder) The *control preorder* $\underset{c}{\searrow}$ over processes is the least preorder closed under structural congruence satisfying the following *control rules*:

$$\begin{array}{cccc}
\bullet P \otimes \bullet Q \underset{c}{\searrow} \bullet(P \otimes Q) & P + Q \underset{c}{\searrow} P & \partial P \underset{c}{\searrow} P & !P \underset{c}{\searrow} P \otimes\, !P \\
(x)\bullet P \underset{c}{\searrow} \bullet(x)P & P + Q \underset{c}{\searrow} Q & \partial P \underset{c}{\searrow} \bullet\partial P & !P \underset{c}{\searrow} \bullet\, !P
\end{array}$$

and such that for all ready $\mathcal{C}$

$$P \underset{c}{\searrow} P' \ \text{ implies } \ \mathcal{C}[P] \underset{c}{\searrow} \mathcal{C}[P'].$$

We use $\overset{1}{\underset{c}{\searrow}}$ to represent a single use of one of the control rules in a ready context; then $\underset{c}{\searrow} = (\overset{1}{\underset{c}{\searrow}})^*$. ■

Each of the six rules for sum, delay and replication either makes some process ready for action, or makes a commitment. But the two rules for committal (the first column) have a different purpose; they lift commitments to the top level, as this example shows:

$$a \bullet P \otimes (x)b \bullet Q \searrow (a \otimes \mathbf{ab}_x b) \bullet (P \otimes (x)Q) \; .$$

6.4 Definition (reduction preorder) The *reduction preorder* $\searrow$ over processes is the least preorder which includes $\underset{a}{\searrow}$ and $\underset{c}{\searrow}$. Thus $\searrow = (\underset{a}{\searrow} \cup \underset{c}{\searrow})^*$. ■

Of course $\searrow = (\overset{1}{\searrow})^*$ where $\overset{1}{\searrow} = (\overset{1}{\underset{a}{\searrow}} \cup \overset{1}{\underset{c}{\searrow}})^*$; so $\searrow$ consists just of a sequence of action and control steps.

A moment's thought shows that an action step creates no new possible control steps; so all the control steps can be done first. Hence

6.5 Proposition $\searrow \; = \; \underset{c}{\searrow} \underset{a}{\searrow}$. ■

One illustration of this result is the example from Synchronous CCS with which we began this section.

To prepare for the introduction of simulation and bisimulation over processes, we now prove two propositions which represent the basic theory of reduction. This theory rests conveniently on the notion of head normal form introduced in the previous section.

6.6 Lemma Let $aH \underset{a}{\searrow} bJ$. Then $a \searrow b$ and $H \equiv J$. (Compare Corollary 5.9.)

Proof It is enough to prove the result for $\overset{1}{\underset{a}{\searrow}}$. In fact it is easiest to prove first that $P \overset{1}{\underset{a}{\searrow}} P'$ implies

$$P \equiv cR, \;\; P' \equiv c'R \;\text{ and }\; c \searrow c' \;\text{ for some }\; c, c' \;\text{and}\; R.$$

We prove this by a straightforward induction on the structure of the ready context $\mathcal{C}$ for which $P \equiv \mathcal{C}[dS]$, $P' \equiv \mathcal{C}[d'S]$ and $d \searrow d'$.

Now take P and P' to be aH and bJ. Then for some c, c', R we have

$$aH \equiv cR, \;\; bJ \equiv c'R \;\text{ and }\; c \searrow c'.$$

Now let R have hnf $c''K$. Then from the uniqueness of hnfs it follows that

$$H \equiv K \equiv J \;\text{ and }\; a = cc'' \searrow c'c'' = b.$$ ■

The corresponding lemma for $\underset{c}{\searrow}$ is easier:

6.7 Lemma Let $aH \underset{c}{\searrow} bJ$. Then there exists c such that $H \underset{c}{\searrow} cJ$ and $ac = b$.

Proof Again, it is enough to prove the result for $\overset{1}{\underset{c}{\searrow}}$. There is some ready context $\mathcal{C}$ and some K, Q such that

$$H \equiv \mathcal{C}[K], \;\; K \overset{1}{\underset{c}{\searrow}} Q \;\text{ and }\; bJ \equiv a\mathcal{C}[Q].$$

Now let $\mathcal{C}[Q]$ have hnf cJ'. Then $H \searrow^1_c cJ'$, and by the uniqueness of hnfs it follows that $b = ac$ and $J \equiv J'$. ■

Putting these two lemmas together, with the help of Proposition 6.5 we have proved a characterisation of reduction in terms of hnfs:

6.8 Proposition Let $aH \searrow bJ$. Then for some c, $H \searrow_c cJ$ and $ac \searrow b$. ■

Our second basic result about dynamics concerns commitments $R \searrow c \bullet R'$. It asserts that commitments of a composite process arise from commitments of its components.

6.9 Proposition Let $P \searrow a \bullet P'$. Then

1. If P is bQ then for some c
$$Q \searrow_c c \bullet P' \text{ and } bc \searrow a.$$
2. If P is $Q \otimes R$, then for some b, c, Q' and R'
$$Q \searrow_c b \bullet Q', \; R \searrow_c c \bullet R', \; b \otimes c \searrow a \text{ and } Q' \otimes R' \equiv P'.$$
3. If P is $(x)Q$ then for some b and Q'
$$Q \searrow_c b \bullet Q', \; \mathbf{ab}_x b \searrow a \text{ and } (x)Q' \equiv P'.$$

Proof For clause 1 consider the hnf of P, hence of R, and apply Proposition 6.8. It will be enough to outline the proof of clause 3; clause 2 is similar. From the assumption, we have $(x)Q \searrow_c a' \bullet P'$ where $a' \searrow a$. Now by inspection the control reduction must take the form of $n \geq 0$ steps of the form
$$(x)Q_i \searrow^1_c (x)Q_{i+1}$$
where $Q \equiv Q_0$, $Q_i \searrow^1_c Q_{i+1}$ $(0 \leq i < n)$ and $Q_n \equiv b \bullet Q'$, followed by a last step of the form
$$(x)Q_n \equiv (\mathbf{ab}_x b)\,(x) \bullet Q' \searrow^1_c (\mathbf{ab}_x b) \bullet (x)Q'.$$
(Note that no further control reduction is possible after this step.) Thus we have $Q \searrow_c b \bullet Q'$ where $\mathbf{ab}_x b = a' \searrow a$ and $(x)Q' \equiv P'$ as required. ■

6.10 Commitment and transition No further control reduction is possible for a commitment
$$P \searrow a \bullet P' \,;$$
if also a is inactive, then no further reduction at all can occur.

At this point recall that we intend to model the single transitions of processes as actions in an action structure. (See the beginning of Section 5.) Commitments are the key to this. As a first approximation we can think of the above commitment as a single transition of P, and write it as $P \xrightarrow{a} P'$, recalling that process calculi like CCS are defined in terms of transitions labelled with actions. But this is not quite enough; for we declared that we should make explicit the way in which an action exerts an effect $e \in E$ upon the continuation of a process. Moreover the transition labels in CCS, except for τ, represent that part of an action which may interact with or be observed by the environment; but we saw at the end of Section 4 that effects are strongly inert, which implies that they cannot react with the environment. So in proceeding from $P \searrow a \bullet P'$ to $P \xrightarrow{a} P'$ we somehow want to absorb any effect of a into P'. How shall we do this? Since an action does not possess an unique effect, though its effects are directed under precomposition (see after Definition 4.5), the following definition is appropriate:

6.11 Definition (transition) The *transition relations* $\{\xrightarrow{a} \mid a \in A\}$ over processes are defined by

$$P \xrightarrow{a} P' \text{ iff, for some } e \text{ and } P'', \quad P \searrow ae \bullet P'' \text{ and } eP'' \equiv P'. \qquad \blacksquare$$

These relations behave well. We now show that they are preserved by composition, abstraction and product, by reaction, and by factorisation of an action through an effect:

6.12 Proposition Let $P \xrightarrow{a} P'$. Then

1. $cP \xrightarrow{ca} P'$;
2. $(x)P \xrightarrow{\mathbf{ab}_x a} (x)P'$;
3. if also $Q \xrightarrow{b} Q'$ then $P \otimes Q \xrightarrow{a \otimes b} P' \otimes Q'$;
4. if $a \searrow a'$ then $P \xrightarrow{a'} P'$;
5. if $a = a'e$ then $P \xrightarrow{a'} eP'$. $\blacksquare$

The next step is to define a notion of (bi)simulation over processes, which uses these transitions. It takes into account that the rôle of an effect e is not to be observed, but to modify the successor process; thus an action with one effect may be simulated by another with different effect, provided the two actions have the same observable content.

6.13 Definition (simulation) A binary relation $\mathcal{S}$ over processes is a *simulation* if, whenever $(P, Q) \in \mathcal{S}$ and $P \xrightarrow{a} P'$, then there exist a', ε and Q' such that

$$a = a'\varepsilon, \quad Q \xrightarrow{a'} Q' \text{ and } (\varepsilon P', Q') \in \mathcal{S}.$$

If $\mathcal{S}$ is also symmetric then it is called a *bisimulation*.

If the pair (P, Q) is in some simulation then we say that Q *simulates* P and write $P \lesssim Q$. If (P, Q) is in some bisimulation then we say that P and Q are *bisimilar* and write $P \sim Q$. $\blacksquare$

We shall hold off the theory of $\lesssim$ and $\sim$ until Section 8, because it is a special case of what is done there. But it is worth recording here a corollary of the main result:

6.14 Theorem (congruence) $\lesssim$ is a precongruence, and $\sim$ a congruence, for the algebra of processes; that is, these relations are respectively a preorder and an equivalence, and are preserved by all the operations introduced in Section 5.1.

Proof This follows from Theorem 8.7 by taking K, the incident-set, to be the entire action structure. $\blacksquare$

It is also worth observing that a simpler notion of simulation suffices, in the case that every action has a pure decomposition (see Definition 4.7):

6.15 Definition (pure simulation) $\mathcal{S}$ is a *pure simulation* if, whenever $(P, Q) \in \mathcal{S}$ and $P \xrightarrow{a} P'$ for some pure a, then there exists Q' such that $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{S}$. If $\mathcal{S}$ is also symmetric, then it is a *pure bisimulation*. ■

Then under the stated condition this gives rise to exactly the same theory:

6.16 Proposition If every action has a pure decomposition, then $P \lesssim Q$ (*resp.* $P \sim Q$) iff (P, Q) is in some pure simulation (*resp.* bisimulation). ■

This theory is quite pleasant. But even if we choose a natural effect structure –and whether or not pure decompositions exist– it turns out that this notion of bisimulation does not yield the familiar equivalence in familiar cases; it is still too strong. We may try to weaken it by restricting the range of a in Definitions 6.13 and 6.15; but this does not always yield a congruence. The remedy lies in the concept of an *incident*, which is the subject of the following section.

7 Incident-sets

We begin this section by showing why bisimilarity, as defined in Definition 6.13, is a stronger equivalence than we would wish.

Recall from Section 3.4 the action structure $\mathsf{SCCS}[Z]$. Choose the effect structure found in Section 4.8, which consists of all actions $(\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle$ whose multiset S is empty. Now $[\,z, \overline{z}\,]$ is a different action from $[\,z', \overline{z'}\,]$. So when we consider processes over this action structure we find

$$[\,z\,] \bullet P \otimes [\,\overline{z}\,] \bullet Q \not\sim [\,z'\,] \bullet P \otimes [\,\overline{z'}\,] \bullet Q,$$

since the former has the transition $\cdots \xrightarrow{[z,\overline{z}]} P \otimes Q$ which cannot be matched by the latter in the sense required for a simulation (6.13). But in Synchronous CCS as originally defined the two processes *are* bisimilar (they are both bisimilar to **id** $\bullet\,(P \otimes Q)$), and we would wish them so.

It is tempting to relax the definition of simulation by restricting consideration to transitions $\xrightarrow{a}$ in which a is inactive, thus excluding things like $[\,z, \overline{z}\,]$. This works for Synchronous CCS, but in general we have to be more careful if we are to retain the congruential property of bisimulation. Indeed, for the synchronous π-calculus (see Part II) it does not work to consider only the inactive actions in simulation.

Throughout this section we assume some fixed action structure A and effect structure E.

7.1 Definition (incident-set) An *incident-set* K is a subset of the actions of A such that

1. For all $a \in A$ and $e \in E$ such that ae is defined, $ae \in K$ iff $a \in K$;
2. For any preaction φ, if $\varphi(a) \searrow \ell \in K$ then there exists $k \in K$ for which $a \searrow k$ and $\varphi(k) \searrow \ell$. ■

Henceforward, when an incident-set K is understood we shall call its members *incidents*, and use k, ℓ, m to range over them. Here, as in the definition of an effect structure, we have imposed the weakest conditions which yield interesting results. For example K need not be a sub-actionstructure.

Clause 2 is the main condition which will ensure congruence. Clause 1 is auxiliary; an important consequence of it is that for a given incident k, its decompositions (k', e) –where k' is also an incident– form a directed set (see after Definitions 4.4 and 4.5):

7.2 Proposition Let $k_1 e_1 = k_2 e_2$. Then there exist k, f_1 and f_2 such that

$$k_1 = kf_1, \quad k_2 = kf_2 \text{ and } f_1 e_1 = f_2 e_2. \qquad \blacksquare$$

Note that this result is only concerned with the statics of incidents.

7.3 Examples (1) It is easy to see that in general, in any action structure A, the entire set A of actions is an incident-set. But we look for smaller incident-sets, because we wish to restrict a in the definition of simulation (6.13) to range over an incident-set. These smaller incident-sets can differ strikingly in different cases.

In both $\mathsf{EXP}[X]$ and $\mathsf{SCCS}[X]$ (Sections 3.3 and 3.5) it turns out that the inactive actions constitute an incident-set. (This appears to be mainly because the reaction relation is confluent in both cases.) For $\mathsf{SCCS}[X]$ these are just the actions $(\vec{x})\,(\boldsymbol{\nu}\vec{y})\,S\,\langle\vec{z}\rangle$ in which S contains no complementary pair $x, \overline{x}$. A smaller incident-set K consists of the inactive actions which also have no restricted particles; that is, if $v \in S$ or $\overline{v} \in S$ then $v \notin \vec{y}$. This is a more appropriate incident-set, since no restricted particle in an action a can ever participate in a reaction of $\varphi(a)$, for any preaction φ. For this K we conjecture that the K-bisimilarity defined in 7.5 below coincides with the original strong congruence for Synchronous CCS [18].

7.4 Examples (2) The case of $\mathsf{CCS}[X]$ is interesting. Recall from Section 3.6 that its reaction rule replaces a complementary pair of particles by the special particle τ, instead of just deleting them. What difference does this make to the possible choice of incident-set? We may take exactly the inactive actions as an incident-set, as for $\mathsf{SCCS}[X]$; this allows an incident to consist of many simultaneous interactions. But due to the presence of the special particle τ we can make the transitions $P \xrightarrow{k} P'$ ($k \in K$) correspond much more closely to the original transitions of CCS [19] which were labelled by a single particle. We outline how to achieve this.

We have referred to the particle x as *positive* and $\overline{x}$ as *negative*; for now, let us say that τ is *both* positive *and* negative. Define the *weight* of an action to be the pair (p, n), where p (resp. n) is the number of its positive (resp. negative) particles; so each τ counts both in p and in n. Then reaction in $\mathsf{CCS}[X]$ preserves the weight of an action; it follows quite easily from this that the set of inactive actions whose weight does not exceed some fixed bound $(p_{\max}, n_{\max})$ constitutes an incident-set. In particular those with weight at most $(1, 1)$ constitute an incident-set; these are the inactive actions with bodies $\emptyset$, $[\,\tau\,]$, $[\,x\,]$, $[\,\overline{x}\,]$ or $[\,x, \overline{y}\,]$ for $x \neq y$. Again, we get a more appropriate incident-set K if we also exclude actions containing restricted particles. This K is in fact the smallest incident-set containing the action $[\,\tau\,]$ (see Theorem 7.7), and the corresponding K-bisimilarity is close to the original strong congruence for CCS; but it is a *stronger* congruence, since it is also preserved by instantiation of port-names. [7]

In our examples so far the natural incidents have been inactive. In general, and especially in the case of the π-calculus and Petri nets, the situation is more delicate; in these cases the inactive actions do not constitute an incident-set, and there are important incident-sets which contain active actions. Incidents for the π-calculus are examined in Part II.

Having looked at some particular cases, we naturally wish to know what non-empty incident-sets to expect in a given action structure A. Can there be none, or many? If there are many, then what can we say about the family of incident-sets in A?

[7] In the standard presentation of CCS [19] instantiation of port-names is not admitted; in fact the strong congruence $p.\mathbf{0} \mid \overline{q}.\mathbf{0} \sim p.\overline{q}.\mathbf{0} + \overline{q}.p.\mathbf{0}$ holds if p and q are different port-names, but fails if they are identical.

We begin with a simple observation. Recall that an incident-set is relative to a given effect structure E. It turns out that, under a modest condition, an incident-set must include the whole of E:

Exercise Let K be an incident-set. Assume that for each $m \in M$ there is a member of K with target arity m. Show that $E \subseteq K$. (*Hint*: use inertia.) ■

Next recall that A is always an incident-set for itself. So in view of our examples, there is often more that one non-empty incident-set. In that case, is there in some sense a best one? We shall show that, given any set B of actions, under a mild condition there is a smallest incident-set which includes B. This is significant for the bisimilarity congruences $\sim_K$ (one for each incident-set K) which we establish in Section 8. For since $\sim_K$ increases as K decreases, we then have the most generous congruential bisimilarity in which two bisimilar processes must at least "agree" with respect to the incidents in B.

For the rest of the section we continue to keep the action structure (A, E) fixed. We shall also assume that $\searrow$ in A is generated by $\overset{1}{\searrow}$ so that $\searrow \; = \; (\overset{1}{\searrow})^*$.

7.5 Definition (convergence) A set $S \subseteq A$ is *convergent* if no infinite reaction sequence $a_0 \overset{1}{\searrow} a_1 \overset{1}{\searrow} a_2 \overset{1}{\searrow} \cdots$ has infinitely many members in S. ■

Now A may not itself be convergent, but it may possess a convergent incident-set. Actually, in all our examples except for Petri nets, A itself is convergent.

If there were a smallest incident-set including B, it should be the intersection of all incident-sets including B. Unfortunately this class is not in general closed under intersection. But:

7.6 Proposition Let $\mathcal{C}$ be any set of incident-sets containing a convergent member. Then $\bigcap \mathcal{C}$ is also an incident-set.

Proof Clearly $\bigcap \mathcal{C}$ satisfies 7.1(1), so we must show that it also satisfies 7.1(2). Suppose not. Then for some preaction φ, some $k \in \bigcap \mathcal{C}$ and some $b \in A$ we have

$$\phi(b) \searrow k \text{ but there is no } \ell \in \bigcap \mathcal{C} \text{ for which } \; b \searrow \ell \text{ and } \varphi(\ell) \searrow k.$$

Take any $K \in \mathcal{C}$. We get a contradiction by finding a reaction sequence

$$b \; = \; b_0 \overset{+}{\searrow} b_1 \overset{+}{\searrow} \cdots$$

which passes infinitely often through K. (Here $\overset{+}{\searrow} \; = \; (\overset{1}{\searrow})^+$, the transitive closure of $\overset{1}{\searrow}$.) We also ensure that $\varphi(b_i) \searrow k$ for all i.

Suppose $b_0, b_1, \ldots, b_j$ are already determined. By assumption, $b_j \notin \bigcap \mathcal{C}$. There are two cases:

1. $b_j \in K$. Then for some other $K' \in \mathcal{C}$, $b_j \notin K'$. Then choose $b_{j+1} \in K'$ such that $b_j \searrow b_{j+1}$ and $\varphi(b_{j+1}) \searrow k$.
2. $b_j \notin K$. Then choose $b_{j+1} \in K$ with the same conditions.

Clearly adjacent members of this sequence are distinct, so $b_j \overset{+}{\searrow} b_{j+1}$ as required; also the sequence passes infinitely often through K by construction. ■

Now, by taking $\mathcal{C}$ in Prop 7.6 to be all incident-sets which include B, we have established the following.

7.7 Theorem (minimum incident-set) Let $B \subseteq A$. If there is a convergent incident-set including B, then there is a smallest incident-set including B. ■

7.8 Discussion The main topic of this section was the structure of the class of incident-sets. The main result here, the existence of a smallest incident-set under certain conditions, does not depend at all upon process structures and the uniform way (defined in Section 5) in which they can be built from action structures. In fact the notion of incident-set appears to be intrinsic to action structure theory. The only part of its definition which is partly motivated by process structure is the notion of preaction, Definition 4.10. However, preactions also seem to have intrinsic interest, being closely connected with inertia.

8 Bisimulation and congruence

We are now ready to introduce a natural class of preorders and equivalences, one for each incident-set. The properties of an incident-set will ensure that they are congruential; this is the main goal of our uniform study of bisimulation.

8.1 Definition (K-simulation) Let K be an incident-set. A binary relation $\mathcal{S}$ over processes is a K*-simulation* if, whenever $(P, Q) \in \mathcal{S}$ and $P \xrightarrow{k} P'$, then there exist k', e and Q' such that

$$k = k'e, \;\; Q \xrightarrow{k'} Q' \;\text{ and }\; (eP', Q') \in \mathcal{S}.$$

If $\mathcal{S}$ is also symmetric, then it is called a K*-bisimulation*.

If (P, Q) is in some K-simulation we say that Q K*-simulates* P, and write $P \lesssim_K Q$. If (P, Q) is in some K-bisimulation we say that P and Q are K*-bisimilar*, and write $P \sim_K Q$. ■

To establish that $P \lesssim_K Q$, or that $P \sim_K Q$, it is convenient to have a more generous class of relations than K-simulations. The following notion is useful:

8.2 Definition (K-simulation up to composition) If $\mathcal{S}$ is a relation over processes, then $\mathcal{S}^{\cdot}$, the *composition closure* of $\mathcal{S}$, is defined by

$$\mathcal{S}^{\cdot} \stackrel{\text{def}}{=} \{(aP, aQ) \mid a \text{ an action and } (P, Q) \in \mathcal{S}\}.$$

$\mathcal{S}$ is a K*-simulation up to composition* if, whenever $(P, Q) \in \mathcal{S}$ and $P \xrightarrow{k} P'$, then there exist k', e, Q' such that

$$k = k'e, \;\; Q \xrightarrow{k'} Q' \;\text{ and } (eP', Q') \in \mathcal{S}^{\cdot}.$$

If $\mathcal{S}$ is also symmetric then it is called a K*-bisimulation up to composition*. ■

The importance of this definition is that, to establish (bi)similarity, we only have to exhibit a (bi)simulation up to composition. To be precise:

8.3 Proposition Let $\mathcal{S}$ be a K-simulation up to composition. Then $\mathcal{S}^{\cdot}$ is a K-simulation. Hence if $(P, Q) \in \mathcal{S}$ then $P \lesssim_K Q$, and if in addition $\mathcal{S}$ is symmetric then $P \sim_K Q$.

Proof It is enough to show the first part. Let $\mathcal{S}$ be a K-simulation up to composition and (aP, aQ) be a typical member of $\mathcal{S}^\cdot$ with $(P, Q) \in \mathcal{S}$. Let $aP \xrightarrow{k} P'$. Then By 7.1 there exist ℓ, e and P'' such that

$$P \xrightarrow{\ell} P'', \;\; a\ell \searrow ke \;\text{ and }\; eP' \equiv P''.$$

Since $(P, Q) \in \mathcal{S}$, there exist ℓ', f and Q'' such that

$$Q \xrightarrow{\ell'} Q'', \;\; \ell' f = \ell \;\text{ and }\; (fP'', Q'') \in \mathcal{S}^\cdot.$$

It follows from Proposition 6.12 that $aQ \xrightarrow{a\ell'} Q''$. But $a\ell' f \searrow ke$ and e is inert, so there exists k' such that

$$a\ell' \searrow k' \;\text{ and }\; k' f = ke.$$

Now from Proposition 7.2 there exist k'', f' and e' such that

$$k' = k'' f', \;\; k = k'' e' \;\text{ and }\; f' f = e' e.$$

So again from Proposition 6.12 we have $aQ \xrightarrow{k'} Q''$, hence

$$aQ \xrightarrow{k''} f' Q'',$$

and since $k = k'' e'$ it only remains to show that $(e'P', f'Q'') \in \mathcal{S}^\cdot$. But

$$(e'P', f'Q'') \equiv (e'eP'', f'Q'') \equiv (f'fP'', f'Q'')$$

and we are done, since $\mathcal{S}^\cdot$ is composition-closed and $(fP'', Q'') \in \mathcal{S}^\cdot$. ∎

As a simple corollary, we find that $\lesssim$ and $\sim$ are preserved by composition:

8.4 Proposition $\;P \lesssim_K Q$ implies $aP \lesssim_K aQ$, and $P \sim_K Q$ implies $aP \sim_K aQ$.

Proof If $P \lesssim_K Q$ then (P, Q) is in some K-simulation $\mathcal{S}$, which is *a fortiori* a K-simulation up to composition. But $(aP, aQ) \in \mathcal{S}^\cdot$ which is itself a K-simulation by the Proposition. Hence $aP \lesssim_K aQ$. ∎

With the help of this we can now establish

8.5 Proposition $\;\lesssim_K$ is a preorder and $\sim_K$ is an equivalence.

Proof It will be enough to show that $\lesssim_K$ (hence similarly $\sim_K$) is transitive. For this, we show that $\mathcal{S} = \{(P, R) \mid \exists Q.\ P \lesssim_K Q \lesssim_K R\}$ is a K-simulation. So let $P \lesssim_K Q \lesssim_K R$ and $P \xrightarrow{k} P'$. It is enough to find k'', f'' and R' for which $k = k'' f''$, $R \xrightarrow{k''} R'$ and $(f''P', R') \in \mathcal{S}$.

First, there exist k', f and Q' for which

$$k = k' f, \;\; Q \xrightarrow{k'} Q' \;\text{ and }\; fP' \lesssim_K Q'.$$

Further, there exist k'', f' and R' for which

$$k' = k'' f', \;\; R \xrightarrow{k''} R' \;\text{ and }\; f'Q' \lesssim_K R'.$$

Now from Proposition 8.4 we know that $f'fP' \lesssim_K f'Q'$; then the required conditions hold if we take $f'' = f'f$. ∎

In final preparation for the proof that these relations are congruential, we shall need a development of Proposition 6.9. It shows how an incident-transition of a composite process must arise from incident-transitions of its components, as follows:

8.6 Proposition Let $P \xrightarrow{k} P'$. Then

1. If P is bQ then for some ℓ and e

$$Q \xrightarrow{\ell} Q', \ \ b\ell \searrow ke \ \text{ and } \ eQ' \equiv P'.$$

2. If P is $Q \otimes R$, then for some ℓ, m, e, Q' and R'

$$Q \xrightarrow{\ell} Q', \ \ R \xrightarrow{m} R', \ \ \ell \otimes m \searrow ke \ \text{ and } \ e(Q' \otimes R') \equiv P'.$$

3. If P is $(x)Q$ then for some ℓ, e and Q'

$$Q \xrightarrow{\ell} Q', \ \ \mathbf{ab}_x\ell \searrow ke \ \text{ and } \ e\,(x)Q' \equiv P'.$$

Proof It will be enough to do clause 2. We have $P \searrow ke \centerdot P''$ for some e and P'' with $eP'' \equiv P'$. So by Proposition 6.9(2) there exist b, c, Q' and R' such that

$$Q \searrow_{c} b \centerdot Q', \ \ R \searrow_{c} c \centerdot R', \ \ b \otimes c \searrow ke \ \text{ and } \ Q' \otimes R' \equiv P''.$$

The rest follows by applying the second clause of Definition 7.1, first to the preaction $(-) \otimes c$ to find ℓ, then to the preaction $\ell \otimes (-)$ to find m. ■

We are now ready for the main result.

8.7 Theorem (incident congruence) K-similarity $\lesssim_K$ is a precongruence, and K-bisimilarity $\sim_K$ is a congruence.

Proof We proved in Proposition 8.4 that composition preserves these relations. We must now do the same for all other process constructions. In many cases this amounts to proving that an appropriate relation is a (bi)simulation up to composition; the detailed reasoning often follows the pattern of the proof of Proposition 8.3.

To show that product preserves $\lesssim_K$ it will be enough to show that

$$\mathcal{S} \stackrel{\text{def}}{=} \{(P \otimes R, \ Q \otimes R) \mid P \lesssim_K Q\}$$

is a K-simulation up to composition.

Let $P \otimes R \xrightarrow{k} S$. Assuming $P \lesssim_K Q$, we shall find k'', e' and T such that $Q \otimes R \xrightarrow{k''} T$, $k''e' = k$ and $(e'S, T) \in \mathcal{S}^{\cdot}$.

First, from Proposition 8.6(2) there exist ℓ, m, e, P' and R' for which

$$P \xrightarrow{\ell} P', \ \ R \xrightarrow{m} R', \ \ \ell \otimes m \searrow ke \ \text{ and } \ e(P' \otimes R') \equiv S.$$

Now $P \lesssim_K Q$, so there exist ℓ', f and Q' such that

$$Q \xrightarrow{\ell'} Q', \ \ \ell' f = \ell \ \text{ and } \ fP' \lesssim_K Q'.$$

We deduce $(\ell' \otimes m)(f \otimes \mathbf{id}) = \ell \otimes m \searrow ke$; but $f \otimes \mathbf{id}$ is inert so there exists k' for which

$$\ell' \otimes m \searrow k' \ \text{ and } \ k'(f \otimes \mathbf{id}) = ke.$$

From the last equation, by Proposition 7.2 there exist k'', e'' and e' for which

$$k' = k''e'', \;\; k = k''e' \;\text{ and }\; e''(f \otimes \mathbf{id}) = e'e.$$

From the above transitions for Q and R, using Proposition 6.12 we deduce

$$Q \otimes R \xrightarrow{k''} e''(Q' \otimes R'),$$

so we shall take T to be the result of this transition. Then from the above equations we find

$$(e'S, T) \;\equiv\; (e''(fP' \otimes R'),\; e''(Q' \otimes R'))$$

which is in $\mathcal{S}^{\cdot}$ since $fP' \lesssim_K Q'$.

This completes the proof that product preserves incident-simulation, and a similar proof works for abstraction. It remains to show that the control operators preserve it. We consider only commitment and replication; summation and delay are no harder.

For commitment, let $P \lesssim_K Q$ and consider $(\bullet P,\; \bullet Q)$. Let $\bullet P \xrightarrow{k} P'$; we shall show $\bullet Q \xrightarrow{k} Q'$ where $P' \lesssim_K Q'$.

We have

$$\bullet P \searrow ke \bullet P'' \;\text{ where }\; eP'' \equiv P'.$$

By Proposition 6.8 it follows that for some c

$$\bullet P \searrow_{c} c \bullet P'' \;\text{ and }\; c \searrow ke.$$

But $\bullet P \searrow^{1}_{c} \ldots$ is impossible, so $c = \mathbf{id}$ and $\bullet P \equiv \bullet P''$, whence $P \equiv P''$ by Proposition 5.9. But $\mathbf{id}$ is inactive, so $ke = \mathbf{id}$. Now $\bullet Q \xrightarrow{\mathbf{1}} Q$, i.e. $\bullet Q \xrightarrow{ke} Q$, so applying Proposition 6.12(5) we get

$$\bullet Q \xrightarrow{k} eQ \;\text{ and }\; P' \equiv eP \lesssim_K eQ.$$

This completes the proof that committal preserves incident-simulation.

For replication, the main task is to show that

$$\mathcal{S} \stackrel{\text{def}}{=} \{(R \otimes\, !P,\; S \otimes\, !Q) \mid R \lesssim_K S,\; P \lesssim_K Q\}$$

is a simulation up to composition – and hence so also is $\mathcal{S} \cup \{(\,!P,\; !Q) \mid P \lesssim_K Q\}$ by a similar argument.

So assume $R \lesssim_K S$ and $P \lesssim_K Q$, and let $R \otimes\, !P \xrightarrow{k} U$. We shall find k'', e' and V for which $S \otimes\, !Q \xrightarrow{k''} V$, $k = k''e'$ and $(e'U, V) \in \mathcal{S}^{\cdot}$. W.l.o.g. we can assume that (for some $n \geq 0$) the first n+1 atomic reductions of the transition are an unfolding of $!P$ into n copies of P followed by a commitment:

$$R \otimes\, !P \searrow^{1}_{c} R \otimes P \otimes\, !P \searrow^{1}_{c} \cdots \searrow^{1}_{c} R \otimes P^n \otimes\, !P \searrow^{1}_{c} R \otimes P^n \otimes \bullet\, !P,$$

where P^n means $\overbrace{P \otimes \cdots \otimes P}^{n\ times}$ (or the term is absent if $n = 0$).

So $R \otimes P^n \otimes \bullet\, !P \xrightarrow{k} U$; then by Propositions 8.6(2) there exist ℓ, m, e, R' and P' such that

$$R \otimes P^n \xrightarrow{\ell} R', \;\; !P \xrightarrow{m} P', \;\; \ell \otimes m \searrow ke \;\text{ and }\; e(R' \otimes P'') \equiv U. \qquad (1)$$

Considering the first of these transitions, we know that $R \otimes P^n \lesssim_K S \otimes Q^n$, so there exist ℓ', f and S' for which

$$S \otimes Q^n \xrightarrow{\ell'} S', \ \ \ell' f = \ell \ \text{ and } \ fR' \lesssim_K S'. \tag{2}$$

Since $(\ell' \otimes m)(f \otimes \mathbf{id}) \searrow ke$ and $f \otimes \mathbf{id}$ is inert, there exists k' for which

$$\ell' \otimes m \searrow k' \ \text{ and } \ k'(f \otimes \mathbf{id}) = ke,$$

whence by Proposition 7.2 there also exist k'', e'' and e' for which

$$k' = k''e'', \ \ k = k''e' \ \text{ and } \ e''(f \otimes \mathbf{id}) = e'e.$$

Considering the second transition of (1), by reasoning similar to the argument for commitment above we find that there exists g such that

$$mg = \mathbf{id} \ \text{ and } \ P' \equiv g\,!P.$$

Now let us consider a transition for $S \otimes \ !Q$; from (2) we easily find that

$$S \otimes \ !Q \xrightarrow{\ell' \otimes \mathbf{1}} S' \otimes \ !Q;$$

also $\ell' \otimes \mathbf{id} = (\ell' \otimes m)(\mathbf{id} \otimes g) \searrow k''e''(\mathbf{id} \otimes g)$, so by Prop 6.12(4),(5) we get

$$S \otimes \ !Q \xrightarrow{k''} e''(\mathbf{id} \otimes g)(S' \otimes \ !Q).$$

Take V to be the result of this transition. Since $k = k''e'$, it only remains to prove that $(e'U, V) \in \mathcal{S}^{\cdot}$. But

$$e'U \equiv e'e(R' \otimes P') \equiv e''(f \otimes \mathbf{id})(R' \otimes g\,!P) \equiv e''(\mathbf{id} \otimes g)(fR' \otimes \ !P),$$

and we are done since $fR' \lesssim_K S'$.

This completes the proof that $\lesssim_K$ is a precongruence; an analogous proof shows that $\sim_K$ is a congruence. ■

Before leaving the topic of K-simulation, we note that when every action has a pure decomposition, then the notion of pure simulation (Definition 6.15) can be extended in an obvious way to *pure K-simulation* and a corresponding equivalence.

9 Related work

This paper is one of many attempts to find a common theoretical framework for concurrent computation. It would take a long essay to classify these attempts, or even to compare them all to this one. I shall mention only three alternative lines.

The *chemical abstract machine* (CHAM) of Berry and Boudol [6] has common aspects with action structures. Indeed, in particular cases such as the π-calculus action structure the actions, with a multiset of particles as their principal component, are close to what the natural CHAM for the π-calculus looks like. Crudely, one can see action structures as an attempt to make an algebra of CHAMs, with parametrisation. Other differences, e.g. in the dynamics, arise from treating reaction as algebraically as possible. Whatever the difference, the CHAM has been a guiding influence in the present work.

Meseguer proposes conditional rewriting logic [16] as a general framework for unifying the study of concurrency. Action structures are not about logic, but they should be compared with the categories, so-called $\mathcal{R}$-systems, which are models –in the technical sense– of a Meseguer rewrite theory. This is a rather general class, and contains the monoidal categories which represent individual Petri nets [17], already referred to in Section 3.8 above. One may contrast this work with action structures as follows: in an $\mathcal{R}$-system the carrier of the algebra consists of the computations of an individual system (e.g. Petri net), while in an action structure the carrier consists of the systems themselves. Direct comparison is therefore difficult; nonetheless it would be of interest to explore a combination of action structures with Meseguer's logical approach.

Recently Nielsen and Winskel have proposed asynchronous transition systems as a general model of concurrency [26]. These are essentially transition systems with additional structure recording the independence among events. This is one difference from action structures, whose definition makes no mention of independence among reactions, though independence can be examined in particular action structures – e.g. in Petri nets. But other differences are perhaps deeper. A good point to examine is the rôle played by morphisms. A morphism in the category of asynchronous transition systems describes, crudely speaking, how one process (e.g. Petri net) may implement or simulate another. In contrast a homomorphism of action structures describes how one *class* of agents, e.g. π-calculus processes, may implement another, e.g. arithmetic expressions. Both notions deserve study; once again comparison is difficult because the models work at different levels.

One may find this disparity among models bewildering. Are we doing something wrong? I think not; concurrent computing can be viewed in so many ways. This has long been apparent at the level of programming languages, and of concrete calculi; it is no surprise to find it so at the more abstract level of general models. Each model must be analysed deeply in its own terms, and be developed with awareness of the others; we must hope that this leads us to unifying concepts, but we should not expect the search to be quick and easy.

PART II: Actions for Pi Calculus

10 Review

In Part I of this paper the theory of action structures was developed, and it was shown how to extend an arbitrary action structure to a process calculus. Part II is devoted to showing how, in particular, an action structure may underlie the π-calculus. We begin by explaining how action structures indeed arose from the need for such a foundation.

10.1 How π-calculus suggests action structures In this subsection we assume nothing about action structures, but rather motivate their introduction. Consider the atomic communications which occur in process calculi, beginning with CCS [19]. The CCS prefix construction $p(y).P$, means "receive any value at port p, bind it to the variable y and proceed to P". The π-calculus has the same construction; but here values, variables and ports are all the same thing: names. Therefore ports can be bound, or parametric; this is what allows the π-calculus to model dynamically changing structure.

Consider the parallel composition

$$S \;\equiv\; x(y).y(u).P \mid \overline{x}z.Q \mid \overline{z}v.R\,;$$

here a communication at port x can occur, yielding the transition $S \longrightarrow S'$ where

$$S' \;\equiv\; z(u).P \mid Q \mid \overline{z}v.R\,.$$

(For simplicity, we assume y not free in P.) At this point a communication at z can occur, which could not occur before; this yields the transition $S' \longrightarrow S''$ where

$$S'' \;\equiv\; \{{}^{v}\!/\!_{u}\}P \mid Q \mid R\,.$$

We can say that the communication at x has *enabled* the one at z, through the instantiation of y to z. This enablement is a phenomenon of the π-calculus, not of CCS.

Prefixing imposes *precedence*, i.e. order of occurrence, among actions in the π-calculus. But since *enablement* is also present, it is reasonable to expect some benefit from separating the two notions, instead of uniting them in a single construction.

Let us look at one way of doing this. Use π to stand for a *particle*, i.e. a simple prefix of the form $x(y)$ or $\overline{x}y$, and consider two cases of the construction $\pi.\pi'.R$:

$$\begin{aligned} P &\;\equiv\; x(y).\overline{z}w.P' \\ Q &\;\equiv\; x(y).\overline{y}w.Q'\,. \end{aligned}$$

In both cases π *precedes* π', while only in Q does π *enable* π'. But suppose that we allow compound prefixes as follows:

$$\begin{aligned} P &\;\equiv\; [\,x(y), \overline{z}w\,].P' \\ Q &\;\equiv\; [\,x(y), \overline{y}w\,].Q' \end{aligned} \tag{3}$$

where the binding of a variable like y is still to the right, but the commutation $\pi, \pi' \equiv \pi', \pi$ is allowed provided it does not make a free variable bound or vice versa. Thus the comma represents enablement.

Such a step has been suggested by several people. Joachim Parrow, who with David Walker and myself first introduced the π-calculus, mooted the idea some years ago. Samson Abramsky found that proof terms for linear logic could be written in π-calculus more accurately with enablement; Gianluigi Bellin and Philip Scott [5] have pursued this idea further. I am also grateful to Bob Constable for pointing out a natural property of concurrent logic programming which can be loosely paraphrased as follows: A variable becomes accessible for a certain purpose just when it is sufficiently instantiated for that purpose. We may think of instantiation as provision of data; in general terms, the idea of data flow as a form of control is an old and natural one, likely to reappear in different contexts.

We now carry our π-calculus example forward a short way, leading to the operations of action structures. Let us suppose that we extend the π-calculus by allowing a prefix to be any sequence of particles, allowing commutations as described above. We may call this the *synchronous* π-calculus, since it corresponds to the π-calculus just as Synchronous CCS [18] corresponds to CCS. Now recall the multiple prefixes in (3); they are

$$a = [\,x(y), \overline{z}w\,] \text{ and } b = [\,x(y), \overline{y}w\,]\,.$$

It is natural to think of a as the juxtaposition, or *product*, of its parts:

$$a = \pi \otimes \pi_z, \text{ where } \pi = x(y) \text{ and } \pi_z = \overline{z}w \; .$$

On the other hand b is not simply a juxtaposition, because one particle enables the other; so we prefer to think of it as a *composition* of its parts:

$$(?) \qquad b = \pi \cdot \pi_y, \text{ where } \pi_y = \overline{y}w.$$

But this formulation pays little attention to how π binds y in π_y, which is the essence of enablement. The most natural thing, rather than introduce a family of binding operators $x(y)$ –one for each pair of names x and y– is to introduce an singly-indexed *abstraction* operator $\mathbf{ab}_y$. This allows a more accurate expression for b:

$$b = b_1 \cdot b_2, \text{ where } b_1 = \mathbf{in}_x \text{ and } b_2 = \mathbf{ab}_y \pi_y.$$

Here $\mathbf{in}_x$ is an action which inputs a name at port x; then the composition ensures that this single name is bound to y in π_y.

This short exercise leaves some loose ends, but it illustrates the rôle played in an action structure by its three operations: *product*, *composition* and *abstraction*. The dynamical ingredient of action structures, *reaction*, is what allows the particles in the action $[\, x(y), \overline{x}z \,]$ to annihilate one another, leaving in their place the substitution $\{^z/_y\}$, which is itself treated as a degenerate kind of action: an *effect*.

10.2 What π-calculus needs from an action structure A uniform way of building a process structure upon an action structure was developed in Sections 5–8 of Part I. We shall now determine what is needed in an action structure, to recover essentially the polyadic π-calculus defined in [20]. The same will suffice also for the synchronous π-calculus.

There are three process constructions in the polyadic π-calculus which are not among those in Section 5.1; they are

$P \mid Q$	parallel composition
$(\boldsymbol{\nu} x)P$	restriction
$\pi.P$	prefixing

where π may be an input particle $x(\vec{y})$, an output particle $\overline{x}\langle\vec{y}\rangle$ or the τ particle (silent action). The names $\vec{y}$ must be distinct in the case of input since they are binding occurrences.

Parallel composition may be defined in terms of other operations, as was done in [18] to derive CCS from synchronous CCS, as follows:

$$P \mid Q \stackrel{\text{def}}{=} (P \otimes \boldsymbol{\partial} Q) + (\boldsymbol{\partial} P \otimes Q) \; .$$

For the rest, we need only require that the π-calculus action structure [8] contain the following five elementary kinds of action:

$\boldsymbol{\tau}$	$: 0 \to 0$	– the silent action
$\mathbf{out}_x$	$: m \to 0$	– output m names through port x
$\mathbf{in}_x$	$: 0 \to m$	– input m names through port x
$\langle x \rangle$	$: 0 \to 1$	– the name x as a datum
$\boldsymbol{\nu}$	$: 0 \to 1$	– a new name as a datum.

[8] We assume now that the arity monoid in this action structure is the natural numbers under addition, and that each name has arity 1.

We shall use the following two abbreviations (where m is the length of $\vec{y}$):

$$\begin{aligned} \langle\vec{y}\rangle &\stackrel{\text{def}}{=} \langle y_1\rangle \otimes \cdots \otimes \langle y_m\rangle \\ (\vec{y})(-) &\stackrel{\text{def}}{=} (y_1)\cdots(y_m)(-) \,. \end{aligned}$$

We can now define prefixing and restriction upon processes as follows, adopting the device of [18] that prefixing incorporates a delay ($\boldsymbol{\partial}$):

$$\begin{aligned} \tau.P &\stackrel{\text{def}}{=} \boldsymbol{\tau}\cdot\bullet\boldsymbol{\partial}P \\ \overline{x}\langle\vec{y}\rangle.P &\stackrel{\text{def}}{=} \langle\vec{y}\rangle\cdot\mathbf{out}_x\cdot\bullet\boldsymbol{\partial}P \\ x(\vec{y}).P &\stackrel{\text{def}}{=} \mathbf{in}_x\cdot(\vec{y})\bullet\boldsymbol{\partial}P \\ (\boldsymbol{\nu}x)P &\stackrel{\text{def}}{=} \boldsymbol{\nu}\cdot(x)P \end{aligned}$$

where for clarity, just for now, we have written composition $\cdot$ explicitly. What is more, the reaction relation $\searrow$ will be the least for which

$$\mathbf{out}_x \otimes \mathbf{in}_x \searrow \boldsymbol{\tau} \otimes \mathbf{id}_m \,.$$

In fact, for the synchronous π-calculus $\boldsymbol{\tau}$ is not needed; for in that case –as for synchronous CCS in Section 3.5– the reaction rule is just

$$\mathbf{out}_x \otimes \mathbf{in}_x \searrow \mathbf{id}_m \,.$$

10.3 Outline of Part II Section 11 presents the action structure PIC_τ for the original π-calculus, together with its effect structure; it also presents PIC, which is a candidate to underlie the synchronous π-calculus. Section 11 finishes with an illustration of synchronous activity, in accessing compound data structures.

In Section 12 an incident-set for PIC_τ is presented; via the general theory of Sections 7–8, this provides a bisimilarity congruence which is satisfactory for the original π-calculus. An incident-set is also presented for the *monadic* PIC. But –surprisingly– it is seen that the analogous set for the full *polyadic* PIC is *not* an incident-set; there is an anomaly.

Section 13 gives a graphical presentation of PIC. This graphical form is a new way of thinking about process mobility, and supplies intuitions which are not seen so clearly in the syntactic form. But it does more; for it suggests a natural way to overcome the anomaly of the previous section.

Section 14 takes up this suggestion; it defines the *reflexive* π-calculus, in terms of the action structure RPIC, which rests upon a new technical notion of *reflexive substitution*. (This notion has the intriguing virtue that it provides a new characterisation for the ubiquitous restriction operator $\boldsymbol{\nu}$; restriction is just the reflexive substitution of a name for itself!) RPIC seems the best candidate to underlie the synchronous π-calculus; it is actually *simpler* than PIC, since it has a purer graphical form; RPIC also yields a natural and elegant incident-set, thereby providing a bisimilarity congruence. This incident-set, and the proof that it is one, rest on a simple graph-theoretic notion of *reachability*.

11 The action structures PIC and PIC_τ

In the previous section we found that the prefixing and restriction operations of the polyadic π-calculus can be defined in terms of a few more elementary kinds of action. We are now ready to examine action structures generated by these primitives (and one other).

The most natural action structure is actually one which underlies synchronous π-calculus. ("Synchronous" here means that arbitrarily many interactions *may* be synchronized; it does not imply that synchronization is *necessary*.) This action structure, which we shall call PIC, needs no silent $\boldsymbol{\tau}$ action. We shall also define PIC_τ, differing from PIC only in the presence of $\boldsymbol{\tau}$ and a corresponding adjustment to the reaction rule; by this means the interleaving treatment of actions in the original π-calculus is faithfully recovered.

11.1 Actions In Section 3.4 the action structure SCCS was defined. The body of each action is a multiset over $\Pi \cup \overline{\Pi}$, where Π and $\overline{\Pi} = \{\overline{\pi} \mid \pi \in \Pi\}$ are respectively the positive and negative particles. Reaction consists simply in cancelling a complementary pair $\pi, \overline{\pi}$. SCCS also allows some particles in an action to be restricted.

To get PIC from SCCS, all we need to do is to allow information, i.e. names, to flow between complementary particles in a reaction. It is also convenient to think of a restriction $\boldsymbol{\nu} x$ as a particle. So particles π of PIC are now given by

$$\pi \ ::= \ x(\vec{y}) \ \Big| \ \overline{x}\langle\vec{y}\rangle \ \Big| \ \boldsymbol{\nu} x .$$

The first two kinds are *input* and *output* (collectively *i/o*) particles, of positive and negative sign respectively. In $x(\vec{y})$ the names $\vec{y}$ must be distinct names, since they are binding. The name x in $\boldsymbol{\nu} x$ is also binding. In PIC_τ, τ is admitted as a fourth kind of particle.

Because of binding, we cannot take the body of an action to be simply a multiset. We wish to allow an action to contain a sequence such as

$$\ldots\ x(y),\ \overline{y}\langle z\rangle\ \ldots$$

representing the receipt of a name y followed by its use as a port. At the process level we may think of the successive reactions of these particles as simultaneous; but we must still admit that the reaction of the second depends upon the receipt of the datum y via the reaction of the first.

We therefore declare the body S of an action to be a *partial sequence*

$$[\,\pi_1,\ \ldots,\ \pi_n\,]$$

of particles; that is, a sequence in which we allow the commutation $\pi, \pi' = \pi', \pi$ of any neighbouring pair neither of which binds a name occurring in the other. The general form of an action $a : m \to n$ is then

$$a \ = \ (\vec{x})S\langle\vec{y}\rangle,$$

where

- $\vec{x}$ is an m-vector of distinct names, the *imported* names;
- S is a partial sequence of particles, the *body*;
- $\vec{y}$ is an n-vector of names, not necessarily distinct, the *exported* names.

The imported names are bound. The scope of each bound name extends to the right of its binding occurrence, and includes the exported names. We identify actions which only differ by alpha-conversion (change of bound names). We tacitly assume in what follows that by suitable alpha-conversion, a name bound in a action is distinct from any other name in that action or in any other action under discussion.

As examples, we immediately define the kinds of action (five for PIC, six for PIC_τ) in terms of which all others may be expressed. They are all listed in Section 10, except $\boldsymbol{\omega}$.

$$
\begin{array}{lrcll}
 & \mathbf{out}_x & : m \to 0 & \stackrel{\text{def}}{=} & (\vec{y})[\,\overline{x}\langle\vec{y}\rangle\,]\langle\,\rangle \qquad (|\vec{y}| = m) \\
 & \mathbf{in}_x & : 0 \to m & \stackrel{\text{def}}{=} & (\,)[\,x(\vec{y})\,]\langle\vec{y}\rangle \qquad (|\vec{y}| = m) \\
 & \langle x\rangle & : 0 \to 1 & \stackrel{\text{def}}{=} & (\,)[\ \ ]\langle x\rangle \\
 & \boldsymbol{\nu} & : 0 \to 1 & \stackrel{\text{def}}{=} & (\,)[\,\boldsymbol{\nu} x\,]\langle x\rangle \\
 & \boldsymbol{\omega} & : 1 \to 0 & \stackrel{\text{def}}{=} & (x)[\ \ ]\langle\,\rangle \\
(\mathrm{PIC}_\tau \text{ only}) & \boldsymbol{\tau} & : 0 \to 0 & \stackrel{\text{def}}{=} & (\,)[\,\tau\,]\langle\,\rangle\ .
\end{array}
$$

We shall often allow ourselves to omit any empty component of an action; e.g. we may write $(\vec{y})[\,\overline{x}\langle\vec{y}\rangle\,]$ for $(\vec{y})[\,\overline{x}\langle\vec{y}\rangle\,]\langle\,\rangle$. We shall also write $\boldsymbol{\nu}\vec{x}$ for a sequence $\boldsymbol{\nu}x_1,\ \boldsymbol{\nu}x_2,\ \cdots$ of particles.

11.2 Operations The action structure operations are defined identically for PIC and PIC_τ. First, the identities are given by

$$\mathbf{id}_m \stackrel{\text{def}}{=} (\vec{x})\langle\vec{x}\rangle \qquad (|\vec{x}| = m).$$

To define the other operations we use substitutions like $\sigma = \{\vec{u}/\vec{v}\}$, where $\vec{v}$ is a vector of distinct names; $\sigma(-)$ is the result of simultaneously replacing each free occurrence of u_i in $(-)$ by the corresponding v_i, first alpha-converting $(-)$ to change any bound uses of $\vec{v}$. We shall also use $,$ to concatenate partial sequences. Suppose $a = (\vec{u})S\langle\vec{v}\rangle$ and $b = (\vec{x})T\langle\vec{y}\rangle$; then composition, product and abstraction are given as follows:

$$
\begin{array}{rcll}
a\cdot b & \stackrel{\text{def}}{=} & (\vec{u})S, \sigma T\langle\sigma\vec{y}\rangle & \qquad (\sigma = \{\vec{v}/\vec{x}\}) \\
a\otimes b & \stackrel{\text{def}}{=} & (\vec{u}\vec{x})S, T\langle\vec{v}\vec{y}\rangle & \\
\mathbf{ab}_x a & \stackrel{\text{def}}{=} & (x\vec{u})S\langle x\vec{v}\rangle. &
\end{array}
$$

Recall our convention that a bound name differs from all names used outside its scope. Thus $a\otimes b$ is simply the juxtaposition of the two actions; no order is dictated by the concatenation, since the convention ensures that $S, T = T, S$ in this case. But in $a\cdot b$ the substitution σ may replace some names free in T by names bound in S, and then $S, \sigma T \neq \sigma T, S$.

It is a routine matter to verify the action structure axioms. Note in particular that the export of x in $\mathbf{ab}_x a$ ensures that $\mathbf{ab}_x$ is functorial.

Exercise Show that $\boldsymbol{\omega}$ cannot be expressed in terms of the other basic actions and the above-defined operations. ■

For future convenience we define the following:

$$
\begin{array}{rcll}
(x)a & \stackrel{\text{def}}{=} & (x\vec{u})S\langle\vec{v}\rangle & \qquad (\text{where } a = (\vec{u})S\langle\vec{v}\rangle,\ x\notin\vec{u}) \\
(\boldsymbol{\nu}x)a & \stackrel{\text{def}}{=} & \boldsymbol{\nu}\cdot(x)a &
\end{array}
$$

11.3 Dynamics The atomic reaction relation $\overset{1}{\searrow}$ of PIC is defined as follows. Whenever a takes the form

$$(\vec{x})\,S, [\,\overline{u}\langle\vec{v}\rangle,\ u(\vec{w})\,], T\,\langle\vec{y}\rangle$$

(after suitable commutations), then

$$a \overset{1}{\searrow} (\vec{x})\, S, \sigma T\, \langle \sigma\vec{y} \rangle,$$

where σ is the substitution $\{\vec{v}/\vec{w}\}$. We call $\overline{u}\langle\vec{v}\rangle, u(\vec{w})$ a *redex* of a.

Example Let $a = (x)[\,\overline{u}\langle x\rangle,\ u(w),\ \overline{w}\langle y\rangle,\ x(z)\,]\langle wz\rangle$. Then

$$\begin{array}{lll} a & \overset{1}{\searrow} & (x)[\,\overline{x}\langle y\rangle,\ x(z)\,]\langle xz\rangle \\ & \overset{1}{\searrow} & (x)\langle xy\rangle. \end{array}$$

■

The port of a redex (in the example, first u then x) may be free or bound. Note that $\overset{1}{\searrow}$ is indeed preserved by the operations; for example $a \overset{1}{\searrow} a'$ implies $b \cdot a \overset{1}{\searrow} b \cdot a'$. The reaction relation $\searrow$ is defined to be $(\overset{1}{\searrow})^*$, the transitive reflexive closure of the atomic relation.

The only difference in PIC_τ is that a redex is not just deleted, but replaced by τ. So in PIC_τ the above example becomes

$$a \overset{1}{\searrow} \cdots \overset{1}{\searrow} (x)[\,\tau,\ \tau\,]\langle xy\rangle\ .$$

Exercise If e is inactive, or even inert, $e \cdot a$ may still have reactions not arising solely from a. Consider the reactions of $\langle u\rangle \cdot a$ in the above example. ■

11.4 Discussion In the original π-calculus every prefix, and therefore every action, consisted of just a single particle. By introducing the action structure PIC or PIC_τ, have we not gone much further than we need to capture the behaviour of the π-calculus?

In fact these action structures reflect the real performance of distributed systems rather more closely than does the original formulation of π-calculus. For the transition system of the latter explicitly prevents more than a single interaction to occur in a single transition, and to implement this would actually require a global scheduler – something which is absent in a distributed system by definition! But the π-calculus was designed to be an tractable theoretical tool, not to be a faithful model of implementation; its interleaving of actions is what makes the expansion law of the π-calculus [24] easy to use, since the number of terms in the expansion does not explode exponentially. The same point was made for CCS in [18]. Moreover, the interleaving was justified at least for CCS [18] (and the same should hold for π-calculus) by a theorem that at the semantic level of observation equivalence it makes no difference whether single observations are treated as simultaneous or interleaved.

In Section 7.4 an appropriate incident-set was defined to recapture the interleaving of actions for CCS; later we shall similarly use PIC_τ to recapture interleaving in the π-calculus.

11.5 Effects and transitions What should be the effect of performing an action in PIC or PIC_τ, i.e. what effect structure should we adopt? There is no single answer; to simplify the decision, we shall consider only actions whose body contains no i/o or τ particles (so they are certainly inert), i.e. actions of the form

$$(\vec{x})[\,\boldsymbol{\nu}\vec{u}\,]\langle\vec{y}\rangle.$$

Into this class fall the following: $\langle x\rangle$, a free datum; $(x)\langle xx\rangle$, which forms a link by identifying two names; and $[\,\boldsymbol{\nu}x\,]\langle xx\rangle$, which forms a private link. These are important cases; we shall see the last illustrated in Section 11.6. But the class contains some undesirables; one such is ω, which discards a name. It is undesirable because discarding can be made into a postcomponent of *any* action; for example if $a : 1 \to 1$ then we can make $(xy)\langle x\rangle$ (which discards its second argument) into an effect of a by writing

$$a \;=\; a \cdot (x)\langle xx\rangle \cdot (xy)\langle x\rangle .$$

To exclude the discarding of names, we propose that the effect structure E for PIC and PIC_τ consist of all actions

$$(\vec{x})[\,\boldsymbol{\nu}\vec{u}\,]\langle\vec{y}\rangle, \qquad \vec{x}\cup\vec{u}\subseteq\vec{y}.$$

It is routine to verify that E is indeed an effect structure. Furthermore, every action has a pure decomposition for E. (See Definition 4.7.) In fact the pure actions P are those actions $(\vec{x})S\langle\vec{y}\rangle$ which satisfy two conditions:

- $\vec{y}$ contains no free names, and no repetitions;
- every restricted name in S occurs in at least one i/o particle.

Exercise Show that every a in PIC or PIC_τ can be expressed as $a = p \cdot e$, where $p \in P$ and $e \in E$. Show further that this is unique up to isomorphism; that is, if also $a = p' \cdot e'$ ($p' \in P$ and $e' \in E$) then there exists $i = (\vec{u})\langle\vec{v}\rangle$, where $\vec{v}$ is a permutation of $\vec{u}$, making the following diagram commute:

$$\begin{tikzcd} & \cdot \arrow[dr, "e"] \arrow[d, "i"] & \\ \cdot \arrow[ur, "p"] \arrow[r, "p'"'] & \cdot \arrow[r, "e'"'] & \cdot \end{tikzcd}$$

This makes (P, E) into a Freyd-Kelly factorization in PIC and in PIC_τ [10], considered as categories. ■

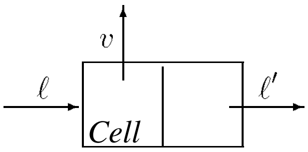
Transitions were introduced in Section 6.11. Recall from there that if P has a commitment $P \searrow a \bullet P'$, then any decomposition (b, e) of a for E can be used to provide a transition

$$P \;\xrightarrow{\;b\;}\; eP'.$$

In the next section we focus on the case when b is a special kind of action known as an *incident*; in such actions, all internal reaction (which could not affect external interaction) has been carried out.

11.6 Illustration Let us now illustrate the power of composite action in PIC, when extended by process structure as described in Section 5. We shall use an encoding of data structures as processes, close to that described in [20]. Specifically, we consider lists. A single list-cell located at ℓ, containing a value-name v and pointing to a successor cell located at ℓ', may be represented by the action

$$[\,\ell(cn),\ \overline{c}\langle v\ell'\rangle\,] : 0 \to 0.$$



We shall represent it by the process *Cell* of arity 3, parametric upon ℓ, v and ℓ':

$$\mathit{Cell} \stackrel{\text{def}}{=} (\ell v \ell')\,\boldsymbol{\partial}[\,\ell(cn),\, \overline{c}\langle v\ell'\rangle\,]\,\boldsymbol{.}\,\mathbf{0}.$$

Similarly, the empty list-cell *Nil* of arity 1, parametric upon its location, is given by

$$\mathit{Nil} \stackrel{\text{def}}{=} (\ell)\,\boldsymbol{\partial}[\,\ell(cn),\, \overline{n}\langle\,\rangle\,]\,\boldsymbol{.}\,\mathbf{0}.$$

For the present discussion we shall understand a *value*, i.e. a list-member, to be a process *Val* of arity 1, of the form $(v)\,!V$; it is parametric upon its location, and the replication allows its repeated "use". Thus $\langle v\rangle \cdot \mathit{Val}$ is a value located at v. So what is the list containing k values $\mathit{Val}_1, \ldots, \mathit{Val}_k$? It is itself a value $L : 1 \to 0$, given by

$$\begin{aligned} L \;=\; (\ell_0)\,!(\boldsymbol{\nu} v_1\ell_1\cdots v_k\ell_k)\Big(&\langle \ell_0 v_1 \ell_1\rangle \mathit{Cell} \otimes \cdots \otimes \langle \ell_{k-1} v_k \ell_k\rangle \mathit{Cell} \otimes \langle \ell_k\rangle \mathit{Nil} \\ &\otimes \langle v_1\rangle \mathit{Val}_1 \otimes \cdots \otimes \langle v_k\rangle \mathit{Val}_k\Big). \end{aligned}$$

Note how each value Val_i has been located for inclusion in the list.

Let us define the single actions *head* and *tail*, which get hold of the first element and the remainder of an arbitrary list value. They are

$$\begin{aligned} \mathit{head} : 1 \to 1 &\stackrel{\text{def}}{=} \mathit{get}\,\mathit{first} \\ \mathit{tail} : 1 \to 1 &\stackrel{\text{def}}{=} \mathit{get}\,\mathit{second}\,, \end{aligned}$$

where

$$\begin{aligned} \mathit{get} : 1 \to 2 &\stackrel{\text{def}}{=} (\ell)[\,\boldsymbol{\nu} cn,\, \overline{\ell}\langle cn\rangle,\, c(v\ell')\,]\langle v\ell'\rangle \\ \mathit{first} : 2 \to 1 &\stackrel{\text{def}}{=} (v\ell)\langle v\rangle \\ \mathit{second} : 2 \to 1 &\stackrel{\text{def}}{=} (v\ell)\langle \ell\rangle\,. \end{aligned}$$

Now let us see how to define processes, given these definitions. Suppose that we have L located at ℓ_0, and we wish to provide its first member as parameter to a process $P : 1$. This can be done by prefixing the action $\langle l_0\rangle \mathit{head}$ to P; then in a single transition this action gains access to the head of L, as follows:

$$\langle \ell_0\rangle \mathit{head}\,\boldsymbol{.}\,P \otimes \langle \ell_0\rangle L \;\xrightarrow{\;b\;}\; (\boldsymbol{\nu} v\cdots)\Big(\langle v\rangle P \otimes \langle v\rangle \mathit{Val}_1 \otimes \langle \ell_0\rangle L\;\cdots\Big).$$

We see that P has indeed "received" v, the location of L's first member Val_1. Thanks to replication, the located list is still available for further access. The action b will normally be an incident (see Section 5); the left-hand side performs a commitment, and the action of this commitment decomposes into (b, e), where the effect e in this case produces the private link v between P and the value Val_1.

Step down the list L need not be performed one-by-one; the following single transition provides access to the fourteenth member of L, where tail^n means the composition of n *tail*s:

$$\langle \ell_0\rangle \mathit{tail}^{13}\mathit{head}\,\boldsymbol{.}\,P \otimes \langle \ell_0\rangle L \;\xrightarrow{\;b'\;}\; (\boldsymbol{\nu} v\cdots)\Big(\langle v\rangle P \otimes \langle v\rangle \mathit{Val}_{14} \otimes \langle \ell_0\rangle L\;\cdots\Big).$$

Let us briefly consider this illustration in broader terms. It naturally prompts the question of why it should be necessary to distinguish process from actions. Why should we not add the control operations $\centerdot$, $\mathbf{0}$, $+$, ∂ and ! to action structures, combining the notions of process structure and action structure?

One answer is that by distinguishing these notions we get a treatment of processes closer to existing process calculi. However, it may be that process calculi themselves would be enriched by treating them as action structures. The second answer is that there are some interesting technical problems to resolve before the extension of action structures (by adding control operations) is well understood. Work in this direction will be reported elsewhere.

12 Incident-sets for π-calculus

In this section we provide incident-sets, as defined in Section 7, both for PIC_τ and for the monadic version of PIC.

12.1 The nature of incidents In many action structures the reaction relation $\searrow$ is confluent. When that occurs, it seems that the interesting incident-sets contain only inactive actions. All the examples of action structures treated in Section 3 were confluent except that for Petri nets, for which an incident-set was not defined. In particular, readers may check that the incident-set given there for CCS contains only inactive actions. (It is also worth noting that it is not a sub-actionstructure; the product of two incidents may indeed be active.)

Now PIC is not confluent, as can be easily seen as follows. Let $a = [\,\overline{x}\langle w\rangle,\ \overline{x}\langle z\rangle,\ x(y)]$; then a has two distinct one-step reactions neither of which can be continued:

$$\begin{array}{ccll} a & \searrow & [\,\overline{x}\langle w\rangle\,] & \not\searrow \\ a & \searrow & [\,\overline{x}\langle z\rangle\,] & \not\searrow \end{array}$$

Indeed we can use this to show that *any* non-empty incident-set for PIC must contain active members. For let K be such an incident-set, and suppose it contains the one-particle action $\ell = [\,\overline{x}\langle z\rangle\,]$. (The exercise below shows that indeed if $K \neq \emptyset$ then $\ell \in K$.) Let $a = [\,\overline{x}\langle z\rangle,\ x(y)]$; a is active, and we show $a \in K$. Let φ be the preaction $[\,\overline{x}\langle w\rangle\,] \otimes (-)$. Then $\varphi(a) \searrow \ell \in K$; so there exists $k \in K$ for which $a \searrow k$ and $\varphi(k) \searrow \ell$. But the only action $a' \neq a$ such that $a \searrow a'$ is the unit action $\mathbf{id}_0$, and clearly $\varphi(\mathbf{id}_0) \not\searrow \ell$. Hence $a \in K$, as we claimed.

Exercise If K is a non-empty incident-set for PIC, show that $[\,\overline{x}\langle z\rangle\,] \in K$. (*Hint*: Let k be any member of K; consider the preaction $k \otimes [\,x(y)] \otimes (-)$. ■

A main aim of this paper is to give interesting and natural incident-sets for PIC. First we shall tackle the easier task of finding a very modest incident-set for PIC_τ.

12.2 Restricted particles and incidents Recall that in a PIC_τ reaction the redex is not merely cancelled, but replaced by τ.

Now let us define the *weight* $|a| = (p, n)$ of an action a exactly as for CCS (Section 7.4); p is the number of input particles, n is the number of output particles, and restriction particles $\boldsymbol{\nu}x$ do not count. We see easily that $|a \cdot b| = |a \otimes b| = |a| + |b|$ and $|\mathbf{ab}_x a| = |a|$. Also, reaction preserves weight; this is the important difference between PIC_τ and PIC, and is what allows PIC_τ to have a rather small incident-set, as we now see.

Call an i/o particle in a is *restricted* if its port is restricted, *free* otherwise. Define the set K_τ^{PIC} to contain every action in PIC_τ with weight at most $(1,1)$ and no restricted i/o particles. We can now establish

12.3 Theorem (incidents for PIC_τ) K_τ^{PIC} is an incident-set for PIC_τ.

Proof It is easy to verify the first clause of Definition 7.1. To verify the second clause, it is enough to verify it for the four basic forms of preaction: $c \cdot (-)$, $c \otimes (-)$, $(-) \otimes c$ and $\mathbf{ab}_x(-)$.

Suppose then that $\varphi(a) \searrow \ell \in K_\tau^{\mathsf{PIC}}$, where φ takes one of these four forms. Note that this reaction takes at most one step. We must find $k \in K_\tau^{\mathsf{PIC}}$ such that $a \searrow k$ and $\varphi(k) \searrow \ell$. We know $|a| \leq |\ell| \leq (1,1)$, so we consider the possible cases for $|a|$.

If $|a| = (0,0)$ then already $a \in K_\tau^{\mathsf{PIC}}$ and we are done. If $|a| = (1,0)$ or $(0,1)$ then a has a single i/o particle. If it is free then $a \in K_\tau^{\mathsf{PIC}}$ and we are done. If not, then $\varphi(a)$ also has this restricted particle, and no other i/o particle with the same port; but then $\varphi(a) \notin K_\tau^{\mathsf{PIC}}$, and a one-step reaction cannot reduce it to a member of K_τ^{PIC}, contrary to hypothesis.

If $|a| = (1,1)$ and $a \notin K_\tau^{\mathsf{PIC}}$ (else we are done) there are two cases:

- a has one free and one restricted particle. Then $\varphi(a)$ also has this restricted particle, and we proceed as in the previous case.

- a has two restricted i/o particles. Then $\varphi(a)$ has the same, and they must form a redex which is reduced in the reaction $\varphi(a) \searrow \ell$. But this reduction can also be done in a, yielding $k \in K_\tau^{\mathsf{PIC}}$ (with a single τ particle) for which $\varphi(k) \searrow \ell$. ∎

This argument would fail if φ could take the form of a postcomposition $(-) \cdot c$; hence the asymmetry in the notion of preaction is important.

We may ask whether this incident-set has special status among those possible for PIC_τ. It was shown in Theorem 7.7 that under certain conditions (satisfied by PIC_τ) there is a smallest incident-set which contains certain given actions. Is K_τ^{PIC} the smallest containing τ? For the full PIC_τ, it is not quite, but nearly so; the discrepancy has to do with i/o particles which carry an empty vector of names. But the assiduous reader may enjoy the following

Exercise Show that in the monadic PIC_τ (where every i/o particle carries a single datum name) K_τ^{PIC} as defined above is the smallest incident-set. *Hint*: By similar methods to the previous exercise, first show that if $\tau \in K$ then every action of weight 1 is in K; then proceed to consider those of weight 2. ∎

How does the bisimilarity induced by K_τ^{PIC} relate to the congruences defined in terms of bisimilarity for the original π-calculus [24]? It turns out to be a little stronger; but it is also more natural in one sense. For it is obtained by a direct definition, whereas the original bisimilarities were not quite congruential; the congruences were defined as "bisimilarity under all substitutions". The relative merits of these congruences remain to be studied; one also hopes that action structures can throw some light upon the structure of the family of strong bisimilarities, not just for π-calculus but more generally.

12.4 Reachability and incidents We now turn to the more challenging question of incident-sets for PIC. Here, the absence of τ means that we no longer have a way to bound the weight of actions in an incident. Instead we rely on another intuition, to reduce the class of possible incidents. Suppose a is in the context of some preaction φ and we are interested in the performance of $\varphi(a)$, e.g. its multistep reduction to an incident. Now a may contribute in two ways to this reduction; it may react internally, or it may interact with φ. Looking at a in isolation, it has two kinds of internal reaction; those which may preclude (or be precluded by) a's possible interactions with its environment, and those which neither preclude nor are precluded by such external interactions.

For the latter kind of internal reaction, nothing (i.e. no external interaction) is lost by letting them be performed earlier rather than later. By confining our attention (in forming an incident-set) to actions which have carried out *all* such internal reactions, we limit the size of the incident-set without endangering the essential second clause of Definition 7.1. A further consideration is that an incident should contain no particle which cannot eventually take part in some reaction.

We are thus led to the notion of *reachability*. In defining it, we assume as usual that all bound names are distinct from each other and from all names occurring free.

Definition The set of names *reachable* in a given action $a \in \mathsf{PIC}$ is the smallest such that

- all the imported and free names of a are reachable;
- if z is reachable and $z(\vec{w})$ or $\overline{z}\langle\vec{w}\rangle$ occurs in a then the names $\vec{w}$ are reachable.

An i/o particle is *reachable* in a if its port is reachable; an action is *reachable* if all its i/o particles are reachable. ■

Example Let

$$a \;=\; (x)[\,\boldsymbol{\nu}u,\ x(y),\ u(v),\ \overline{u}\langle y\rangle,\ \overline{v}\langle z\rangle\,]\langle yu\rangle;$$

then only the names x, y and the particle $x(y)$ are reachable. Note that a contains a redex $u(v)\,\overline{u}\langle y\rangle$ which has an unreachable port. If we reduce it we have

$$a \searrow^{1} (x)[\,\boldsymbol{\nu}u,\ x(y),\ \overline{y}\langle z\rangle\,]\langle yu\rangle;$$

the reaction has created a reachable particle $\overline{y}\langle z\rangle$ from one which was previously unreachable, and has reduced the unreachable action a to a reachable one. ■

Surprisingly enough, the status of reachable actions differs according to whether we are considering full PIC, or only monadic PIC. We can prove the following:

12.5 Theorem (incidents for monadic PIC) The set K^{PIC} of reachable actions is an incident-set in monadic PIC. ■

The proof of the theorem follows similar lines to that of Theorem 14.4, concerning an incident-set for the reflexive π-calculus. I conjecture, but have not proved, that K^{PIC} is the smallest incident-set in monadic PIC which includes the effects.

In Section 12.5 below we see why K^{PIC} is *not* an incident-set for full PIC. This anomaly arises from a phenomenon of the binding convention in PIC. It may seem puzzling, if we recall from [20] that the polyadic π-calculus is definable from the monadic calculus. But that definition relies upon the prefixing construction of the π-calculus, and prefixing is absent in PIC! In Section 14 below we enrich PIC to remove the anomaly.

Some insight into incident-sets can be gained from the following exercise, which shows clearly how the notion of incident separates proper from improper transitions.

Exercise Consider again the list-processing illustration in Section 11.6. Consider the commitments of $(\boldsymbol{\nu}\ell_0)(\,\langle\ell_0\rangle head \bullet P \otimes \langle\ell_0\rangle L\,)$, where now the access to the list is restricted. Show that there is a large number of possible commitments of the form

$$(\boldsymbol{\nu}\ell_0)\Big(\,\langle\ell_0\rangle head \bullet P \otimes \langle\ell_0\rangle L\,\Big) \searrow a \bullet Q,$$

but only one in which a an incident. Show that this incident decomposes into two actions; the incident $[\,\boldsymbol{\nu}cn\,]$ which discards some "used" names, and the effect

$$[\,\boldsymbol{\nu}\ell_0,\ \boldsymbol{\nu}v_1\ell_1 \cdots v_k\ell_k\,]\langle\ell_0 v_1 v_1\ell_1 \cdots v_k\ell_k\rangle$$

which transmits private names to Q. ■

12.6 Anomaly In PIC, though not in monadic PIC, the interplay between binding and reaction has a surprising consequence which can be illustrated by a simple example. Let

$$b \;=\; [\,v(x),\ \underline{\overline{u}\langle xy\rangle,\ u(zw)},\ \overline{v}\langle w\rangle\,]\langle x\rangle,$$

or by rearrangement

$$b \;=\; [\,u(zw),\ \underline{\overline{v}\langle w\rangle,\ v(x)},\ \overline{u}\langle xy\rangle\,]\langle x\rangle,$$

where the two redexes in b are shown by underlining. These two redexes are disjoint (i.e. they involve distinct particles), so one may expect that they can be reduced in arbitrary order. But here are the two reaction sequences for b:

$$\begin{array}{lll} b & \searrow & [\,\underline{v(x),\ \overline{v}\langle y\rangle}\,]\langle x\rangle \searrow \langle y\rangle \\ b & \searrow & [\,u(zw),\ \overline{u}\langle wy\rangle\,]\langle w\rangle \ \ \not\searrow^{1} . \end{array}$$

The second sequence can go no further! There is no redex, due to PIC's binding convention. Yet if we allowed a reaction, then the result should again be $\{y/w\}\langle w\rangle = \langle y\rangle$. In forbidding such redexes, PIC denies certain reactions which are intuitively acceptable. [9]

Our example can now be used to show that K^{PIC} is indeed not an incident-set for full PIC. Let c be a slight variant of b, as follows:

$$c \;=\; [\,\boldsymbol{\nu}v,\ v(x),\ \underline{\overline{u}\langle xy\rangle,\ u'(zw)},\ \overline{v}\langle w\rangle\,]\ .$$

Now let φ be the preaction $\langle u\rangle\mathbf{ab}_{u'}(-)$; then (as in the first reaction sequence for b)

$$\varphi(c) \searrow [\,\boldsymbol{\nu}v\,]\langle u\rangle\ .$$

Since $[\,\boldsymbol{\nu}v\,]\langle u\rangle \in K^{\mathsf{PIC}}$, we look for $k \in K^{\mathsf{PIC}}$ such that

$$c \searrow k \ \text{ and } \ \varphi(k) \searrow [\,\boldsymbol{\nu}v\,]\langle u\rangle.$$

But $c \notin K^{\mathsf{PIC}}$, and its only reaction is

$$c \searrow c' \ \ \not\searrow^{1}, \ \ \text{where } c' \;=\; [\,\boldsymbol{\nu}v,\ u'(zw),\ \overline{u}\langle wy\rangle\,]\ ;$$

furthermore $\varphi(c') = [\,\boldsymbol{\nu}v,\ u(zw),\ \overline{u}\langle wy\rangle\,]\langle u\rangle$ has no redex, so cannot reduce to $[\,\boldsymbol{\nu}v\,]\langle u\rangle$. So K^{PIC} is not an incident-set for full PIC.

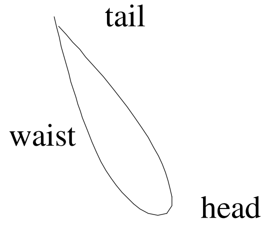
This anomaly of resolved in Section 14.2 by the notion of *reflexive substitution*.

[9] Even in monadic PIC, an action may have two disjoint redexes, but after one is reduced the other is no longer a redex; consider $[\,v(x), \overline{u}\langle x\rangle, u(y), \overline{v}\langle y\rangle\,]$. But in monadic PIC it turns out that if both can be reduced in *one* order, then both can be reduced in the other order. This is what lies behind Theorem 12.5, and fails for polyadic PIC.

13 A graphical action structure

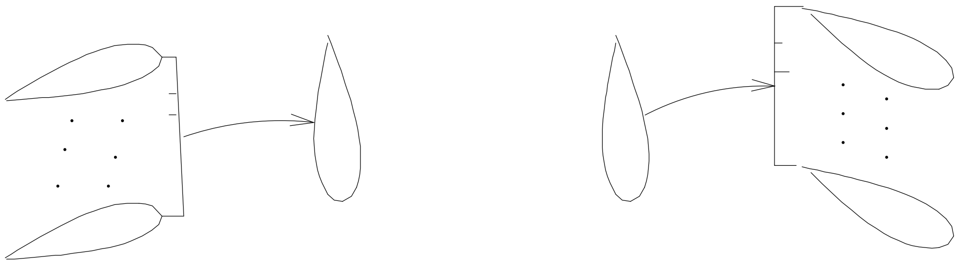
The action structure PIC has been presented in a very syntactic way. This helps formal manipulation, but some essential aspects are still hidden. Here we show that both the statics and the dynamics of PIC correspond to simple graphical ideas. This graphical presentation also helps us to understand the notions of effect and incident. Further, it suggests an enrichment of PIC which has appealing properties.

13.1 Bare π-nets We begin with a simple kind of graph called a *bare π-net*. It has *nodes* and *directed arcs*. Each node is drawn as a pointed oval with a *head*, a *waist* and a *tail* as shown.



Each directed arc is either a *source* or a *target* arc. We first consider *monadic* arcs. A monadic source arc is drawn from the head of a node (the *source* of the arc) to the waist of a node (the *port*); a monadic target arc is drawn from the waist of a node (the *port* of the arc) to the tail of a node (the *target*).

In general, an arc may be *polyadic*; a source arc may have an ordered sequence of k source nodes, and a target arc may have an ordered sequence of k distinct target nodes ($k \geq 0$):



(Note that the arc exists even when $k = 0$.)

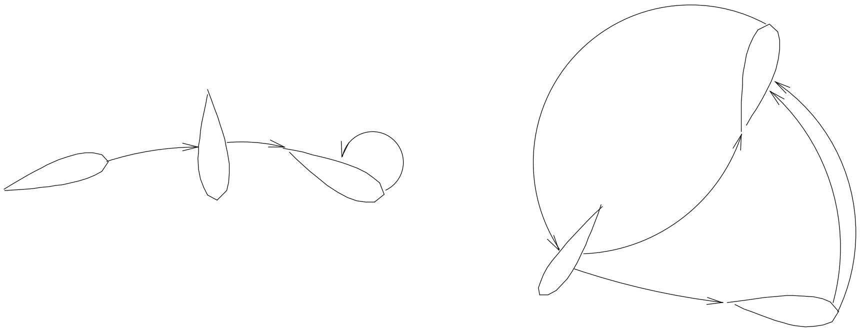
In what follows we use m, n, p to stand for nodes. A sequence $m_0, \ldots, m_k$ of nodes ($k \geq 0$) is a *target path* if for each i ($0 \leq i < k$) there is a target arc from m_i to m_{i+1}. If $k > 0$ and $m_k = m_0$ then the path is a *target cycle*. A bare π-net is subject to the following constraints:

- Each node is a target of at most one arc;

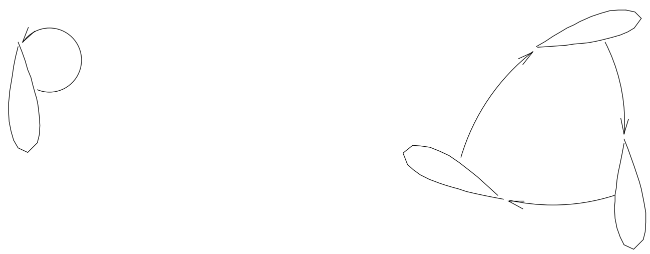
- There are no target cycles.

We say a (bare) π-net is *monadic* if all its arcs are monadic. We shall use only monadic π-nets in our illustrations, but the work of this section and the next applies to all π-nets.

Here are two admissible bare monadic π-nets:



Here are two which are inadmissible, because they have (in fact they are) target cycles:



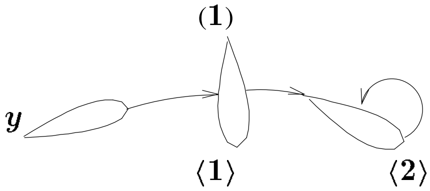
If a node m is the port of a target arc then we say it *carries* the arc and its target nodes; similarly, if m is the port of a source arc then it carries the arc and its source nodes.

13.2 Actions and π-nets We now define π-nets, and show how to represent the actions of PIC as π-nets.

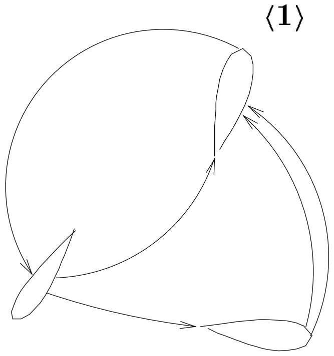
Suppose that Z is the set of names free in an action $a : h \to k$ of PIC. Then the π-net representing a consists of a certain bare π-net together with

- an injective map from $Z \cup \{1, \ldots, h\}$ into the non-target nodes;
- a map (not necessarily injective) from $\{1, \ldots, k\}$ into the nodes.

Here are two π-nets, obtained by adding appropriate maps to our two admissible bare π-nets. The first map is indicated by *tail-labels* such as $\boldsymbol{y}$ or $(\mathbf{1})$, the second by *head-labels* such as $\langle \mathbf{1} \rangle$. Note that our definition ensures that a node may not bear a tail-label if it is a target of an arc.



$$\boldsymbol{a : 1 \to 2 \;=\; (x)[\,\overline{x}\langle y\rangle,\; x(z),\; \overline{z}\langle z\rangle\,]\langle xz\rangle}$$



$$\boldsymbol{b} : \boldsymbol{0} \rightarrow \boldsymbol{3} \;=\; [\, \boldsymbol{\nu x},\ \boldsymbol{x(y)},\ \overline{\boldsymbol{x}}\langle \boldsymbol{y} \rangle,\ \boldsymbol{x(z)},\ \overline{\boldsymbol{y}}\langle \boldsymbol{z} \rangle,\ \overline{\boldsymbol{y}}\langle \boldsymbol{z} \rangle \,]\langle \boldsymbol{yzz} \rangle$$

For each of these π-nets we have shown the PIC action which it stands for. To translate an arbitrary PIC action a to a π-net is quite simple. Let $a = (\vec{x})S\langle\vec{y}\rangle : h \rightarrow k$. Then:

- create a node $\boldsymbol{z}$ (i.e. having this tail-label) for each name z occurring free or bound in a;
- for $1 \leq j \leq k$ give node $\boldsymbol{y}_j$ the head-label $\langle \boldsymbol{j} \rangle$;
- for each particle $\overline{u}\langle\vec{v}\rangle$ in S draw a source arc with source nodes $\vec{\boldsymbol{v}}$ and port node $\boldsymbol{u}$;
- for each particle $u(\vec{v})$ in S draw a target arc with port node $\boldsymbol{u}$ and target nodes $\vec{\boldsymbol{v}}$, removing the latter tail-labels;
- for each particle $\boldsymbol{\nu} u$ in S remove the tail-label of node $\boldsymbol{u}$;
- for $1 \leq i \leq h$, replace the tail-label of node $\boldsymbol{x}_i$ by $\langle \boldsymbol{i} \rangle$.

The result, which we call *draw*(a), is clearly a π-net on the names free in a. In particular it has no target cycles because a contains no cycle of input particles each binding the next.

It is also easy to define an inverse translation

$$\textit{write} : \pi\text{-nets} \rightarrow \mathsf{PIC};$$

to form *write*(N), first introduce a new name z for every unnamed node in N; then construct the particles – one i/o particle for each arc, and one restriction particle for each non-target node with no tail-label. We omit the remaining details, since they are obvious. The absence of target cycles ensures that the particles in *write*(N) can be written in an appropriate linear order.

Now it is easy to show that *write*(*draw*$(a)) = a$. It is nearly true that *draw*(*write*$(N)) = N$; the only discrepancy is when N contains a node which has a name $\boldsymbol{z}$ (say) as tail-label, but bears no arcs and no head-label. Let us call such a node *vacuous*. A vacuous node has no significance; it affects neither the algebraic operations nor the dynamics (both defined below). So we shall not distinguish two π-nets differing only in their vacuous nodes. Then indeed we have a bijection

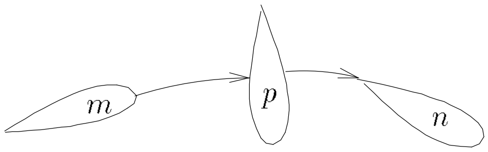
$$\mathsf{PIC} \;\underset{\textit{write}}{\overset{\textit{draw}}{\rightleftarrows}}\; \pi\text{-nets}\,.$$

13.3 Operations and dynamics The operations of composition, product and abstraction on π-nets are simply described. The first two involve the *coalescing* of two nodes; this means forming a single node bearing the arcs and labels of both the given nodes. Whenever this is done, it is ensured that the resulting π-net is still legal.

Let $M_1 : h_1 \to k_1$ and $M_2 : h_2 \to k_2$ be π-nets.

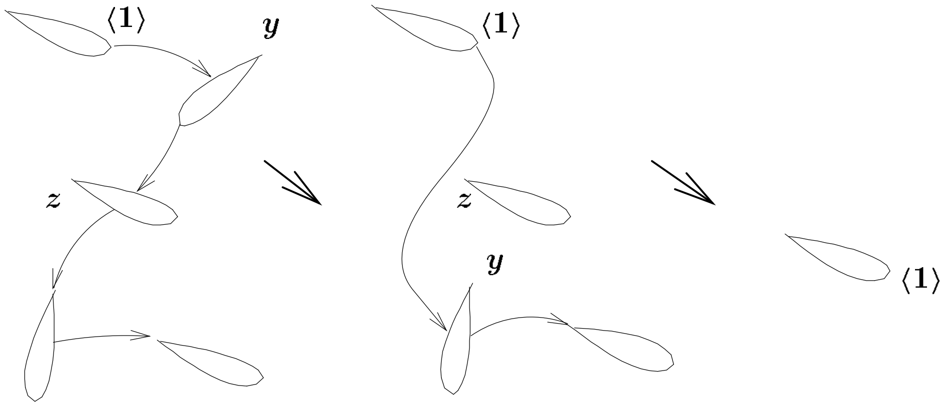
- To form $M_1 \cdot M_2$, for each i ($1 \le i \le k_1 = h_2$) coalesce the node in M_1 whose head-label is $\langle \boldsymbol{i} \rangle$ with the node in M_2 whose tail-label is $(\boldsymbol{i})$, and remove these labels. Also coalesce nodes in M_1 and in M_2 which share a name $\boldsymbol{z}$ as tail-label.
- To form $M_1 \otimes M_2$, first increment all tail-labels $(\boldsymbol{i})$ and head-labels $\langle \boldsymbol{j} \rangle$ in M_2 by h_1 and k_1 respectively. Also, as above, coalesce any pair of nodes with the same tail-label $\boldsymbol{z}$.
- To form $\mathbf{ab}_x M_1$, first increment all tail-labels $(\boldsymbol{i})$ and head-labels $\langle \boldsymbol{j} \rangle$ in M_1 by 1. Then assign head-label $\langle \mathbf{1} \rangle$ to the node with tail-label $\boldsymbol{x}$, and replace its tail-label by $(\mathbf{1})$.

We now define reaction. in the monadic case a *redex* of M consists of a configuration



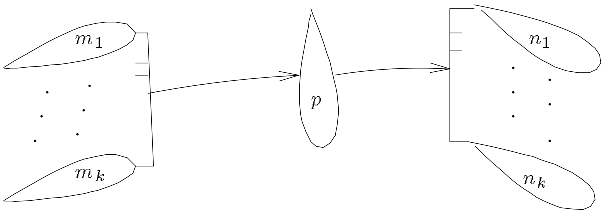
where there is no target path from n to m. (This implies $m \neq n$.) Then there is a one-step reduction $M \overset{1}{\searrow} M'$, where M' is derived from M just by removing the two arcs shown and coalescing m with n. We call m, n and p respectively the *source*, the *target* and the *port* of the redex. The reaction relation $\searrow$ π-nets is defined to be the transitive reflexive closure of $\overset{1}{\searrow}$. Note that reaction does not depend at all upon labelling; in effect we have defined it for bare π-nets.

Here is an illustration of a two-step reaction:



$$[\,\boldsymbol{\nu x},\ \overline{\boldsymbol{y}}\langle \boldsymbol{x} \rangle,\ \overline{\boldsymbol{z}}\langle \boldsymbol{y} \rangle,\ \boldsymbol{z}(\boldsymbol{u}),\ \boldsymbol{u}(\boldsymbol{v})\,]\langle \boldsymbol{x} \rangle \searrow [\,\boldsymbol{\nu x},\ \overline{\boldsymbol{y}}\langle \boldsymbol{x} \rangle,\ \boldsymbol{y}(\boldsymbol{v})\,]\langle \boldsymbol{x} \rangle \searrow [\,\boldsymbol{\nu x}\,]\langle \boldsymbol{x} \rangle$$

In the polyadic case a redex is of the following form, with no target path from any n_j to any m_i:



The reduction again consists of removing the two arcs and coalescing m_i with n_i, for each i. (Note that reduction is not vacuous when $k = 0$, since the arcs are removed.)

This completes the definition of π-nets as an action structure. It is now a routine matter to check that the bijection consisting of the two translations *draw* and *write* is actually an isomorphism of action structures; that is, it preserves both the operations and the dynamics in both directions. Thus π-nets are an alternative presentation of PIC.

Recall that the effect structure of PIC consists of all actions of the form $(\vec{x})[\,\boldsymbol{\nu}\vec{y}\,]\langle\vec{z}\rangle$ where $\vec{y} \cup \vec{z} \subseteq \vec{x}$. In terms of π-nets, these are just arc-free nets where every node has a head-label.

13.4 Reflexive π-nets Having seen that the action structure of π-nets is isomorphic to PIC, one naturally asks what is gained by this new presentation. One answer is that it gives a less formal, more geometric, view of reaction. We may compare it with the geometry of interaction of Lafont [15], following Girard [11]; it differs notably in that reaction in PIC is not a confluent reduction relation. Another answer is that the geometric view may suggest generalisations or variations of PIC. We now examine one generalisation.

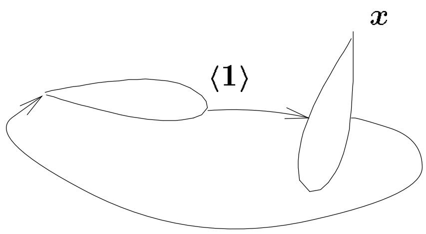
Two related constraints have been placed upon π-nets:

C1 A π-net must have no target cycles.

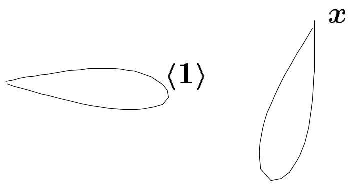
C2 In a redex there must be no target path from a target node to a source node.

C1 arises from the binding convention in PIC. C2 may also be considered to arise from binding. First, it disallows $x(\vec{y}), \overline{x}\langle\vec{z}\rangle$ as a redex if $\vec{y}$ and $\vec{z}$ have any name in common. In reducing a redex such as $x(y), \overline{x}\langle y\rangle$ apparently the two particles would simply to cancel each other, causing the name y – previously bound – to become free; at first sight it is unclear what this could mean. Second, C2 prevents a reaction from creating a target cycle and thus violating C1.

We now define *reflexive* π-nets to be the more general class of nets for which constraints C1 and C2 are removed. The operations and reaction relation for reflexive π-nets are defined exactly as before, but without the constraints. It is routine to check that this gives us another action structure, of which the (non-reflexive) π-nets form a sub-actionstructure. We shall shortly give it a syntactic formulation, and it will be called RPIC. But first let us focus on its most interesting new aspect. Consider the following (non-reflexive) π-net:



It is the action $[\, x(y),\ \overline{x}\langle y\rangle\,]\langle y\rangle$, and in PIC it has no reaction; but under the reaction relation which we have defined over reflexive π-nets it reduces to



The port node x has become vacuous, so the result is the action $[\,\boldsymbol{\nu} y\,]\langle y\rangle$, or just $\boldsymbol{\nu}$. So apparently in RPIC, just as the reaction of $x(y), \overline{x}\langle z\rangle$ produces the substitution $\{z/y\}$ when $z \neq y$, so the reaction of $x(y)\,\overline{x}\langle y\rangle$ produces the restriction $\boldsymbol{\nu} y$. This reaction is geometrically sensible, since it arises simply from removing a constraint previously placed upon the reaction rule; but its implication is remarkable, for it suggests that restriction is formally a kind of *self*-substitution (but a kind which is unfamiliar). The substitution $\{z/y\}$ means that y should be taken to denote z; so if we follow the geometry's suggestion then the self-substitution $\{y/y\}$ should mean that y denotes itself, i.e. *denotes nothing else*. Though fanciful, this aligns with the intuition that a restricted name is indeed distinct from every other name. It explains why we have used the term *reflexive* for the wider class of nets.

In the next section we set out to make self-substitution not only geometrically sensible but also formally sensible.

14 Reflexive π-calculus

In this section we put together two strands of the development so far. In Section 12 we studied incident-sets for PIC_τ and PIC; in the latter case we found an incident-set for the monadic sub-actionstructure of PIC, but met a difficulty with the full structure. In Section 13 the graphical characterisation of PIC suggested a richer action structure, reflexive PIC or RPIC, which does not arise from the formal presentation.

Here we present RPIC formally, a surprisingly easy task once we establish the idea of reflexive substitution. Then we find a natural and elegant incident-set for RPIC.

14.1 The action structure RPIC: statics An action of RPIC takes the form

$$(\vec{x})M\langle\vec{y}\rangle,$$

where the body M is a *multiset* of particles, in contrast to PIC where the body is a *partial sequence* of particles. (The particles are unchanged.) Associated with this difference is a difference of binding convention. In RPIC, as in PIC, there is just one binding occurrence of each bound name x in an action a; but the scope of x is now the whole of a, whereas in PIC it extends only to the right of the binding occurrence.

The effect is to admit cyclic binding dependency among particles (i.e. target cycles). Thus the actions

$$[\, x(y),\ y(z),\ z(x)\,] \ \text{ and } \ [\, x(x)\,]$$

are admissible in RPIC. (Note that neither of them has any free names!)

At present we seek no direct significance for these target cycles; they should be treated for now as a formal extension of PIC which allows simple technical development. For one thing, multisets are easier to handle that partial sequences; for another, there will be (as we said) a natural incident-set – and this arises because RPIC does not suffer from the anomaly of PIC's binding rule which was pointed out in Section 12.6.

The operations of composition, product and abstraction are defined as for PIC, but with multiset union in place of the concatenation of partial sequences. If $a = (\vec{u})M\langle\vec{v}\rangle$ and $b = (\vec{x})N\langle\vec{y}\rangle$ then

$$\begin{array}{rcll} a \cdot b & \stackrel{\text{def}}{=} & (\vec{u})\, M \cup \sigma N\, \langle\sigma\vec{y}\rangle & \qquad (\sigma = \{\vec{v}/\vec{x}\}) \\ a \otimes b & \stackrel{\text{def}}{=} & (\vec{u}\vec{x})\, M \cup N\, \langle\vec{v}\vec{y}\rangle & \\ \mathbf{ab}_x a & \stackrel{\text{def}}{=} & (x\vec{u})\, M\, \langle x\vec{v}\rangle . & \end{array}$$

14.2 The action structure RPIC: dynamics In RPIC any complementary pair of particles

$$\overline{u}\langle\vec{v}\rangle,\ u(\vec{w})$$

in the body of an action a is a redex of a; the contrast with PIC is that members of $\vec{v}$ may be among the bound names $\vec{w}$. By analogy with PIC we allow the one-step reduction

$$a \searrow^{1} \{\vec{v}/\vec{w}\} * \widehat{a},$$

where $*$ represents the action of *reflexive substitution* (of $\vec{v}$ for $\vec{w}$) and $\widehat{a}$ stands for a with the redex deleted. Informally, reflexive substitution performs literal replacement of names, but introduces a restriction particle whenever a name is required (via a cycle) to "replace itself".

In defining the operation $\{\vec{y}/\vec{x}\} * a$ we assume always that the names $\vec{x}$ are distinct and not bound in a. But $\vec{y}$ may include members of $\vec{x}$ and even bound names of a; reflexive substitution operates upon a fixed presentation of an action, with no alpha-conversion.

First, define ${}^{y/x}(-)$ to mean the literal replacement of the name y for the name x; for example ${}^{y/x}((y)\langle x\rangle) = (y)\langle y\rangle$. Then reflexive substitution is given recursively by

$$\begin{array}{rcl} \{\,\} * a & \stackrel{\text{def}}{=} & a \\ \{z/w, \vec{y}/\vec{x}\} * a & \stackrel{\text{def}}{=} & \left\{ \begin{array}{ll} \{{}^{z/w}\vec{y}\,/\vec{x}\} * {}^{z/w}a & (z \neq w) \\ \{\vec{y}/\vec{x}\} * (\boldsymbol{\nu} w)a & (z = w) \end{array} \right. \end{array}$$

To ensure that this is a proper definition, it is important to check that it yields the same result (up to alpha-conversion) no matter in which order the elements of a substitution $\{\vec{y}/\vec{x}\}$ are applied. This can be proved by induction on the size of the substitution. The reader may check, for example, that

$$\{y/x, x/y\} * [\,\overline{y}\langle z\rangle\,]\langle x\rangle \;=\; \{y/y\} * [\,\overline{y}\langle z\rangle\,]\langle y\rangle \;=\; [\,\boldsymbol{\nu} y,\, \overline{y}\langle z\rangle\,]\langle y\rangle,$$

and that a different substitution order yields the same up to alpha-conversion:

$$\{x/y, y/x\} * [\,\overline{y}\langle z\rangle\,]\langle x\rangle \;=\; \{x/x\} * [\,\overline{x}\langle z\rangle\,]\langle x\rangle \;=\; [\,\boldsymbol{\nu} x,\, \overline{x}\langle z\rangle\,]\langle x\rangle\ .$$

14.3 Discussion It is a routine matter now to verify that the action structure of reflexive π-nets and the action structure RPIC are isomorphic. The isomorphism is just as described in Section 13.2 for the non-reflexive case. It remains to be seen, by experimenting with applications, whether modelling with RPIC is significantly different from PIC in cases where the richer constructions of RPIC as not needed. Indeed, it is not even clear when and where this extra richness will be useful in practice. What then are the advantages of RPIC?

First, though its significance is not clear, we have found a new characterization of restriction; the reduction

$$[\, u(x),\ \overline{u}\langle x\rangle \,]\langle x\rangle \searrow \boldsymbol{\nu}$$

shows it arising from an interaction, rather than as an independent notion.

Second, the theory becomes easier in at least one important matter. For not only do we now find a natural incident-set where we failed for PIC, but we also find this easier to verify than in the case of monadic PIC because we can use graph-theoretic methods without being hampered by awkward constraints.

We now set about this. First, we rephrase the notion of reachability (Section 12.4) in graphical terms. The *reachable* nodes and arcs of a reflexive π-net are the least set such that

- every node with a target-label is reachable;

- every node and arc carried by a reachable node is reachable.

(This is a special case of a more general definition given later.) Further, a reflexive π-net is *reachable* if all its arcs are reachable.

The remainder of this section is devoted to a proof of the following

14.4 Theorem (incidents for RPIC) The set K^{RPIC} of reachable reflexive π-nets is an incident-set for RPIC. ■

We need some definitions and lemmas to prove this result.

14.5 Redexes and residuals A *redex* of $a \in \mathsf{RPIC}$ consists of a source arc and a target arc with a common port, and their associated source and target nodes. This is just as in Section 6.3 for π-nets, but without constraints. Two redexes are *disjoint* if their source arcs differ and their target arcs differ.

We use r, s, t to stand for redexes. The one-step reaction or *firing* corresponding to a redex r is written

$$a \overset{r}{\searrow} a^r,$$

where the *reductum* a^r is gained from a by removing the source and target arcs of r and coalescing each source node with the corresponding target node.

Given a redex r of a, each node n in a corresponds in an obvious way to a node n^r in a^r called its *residual* (under r). Thus if m and n are a corresponding source and target of the redex r we have $m^r{=}n^r$, but otherwise distinct nodes have distinct residuals. Similarly each arc π in a except the source and target arcs of r has a residual arc π^r in a^r; distinct arcs always have distinct residuals. A collection of arcs and nodes in a, such as a redex, also has a residual under r (the collection of residuals of its members). The following is easy to prove in RPIC (but is false in PIC, as shown in Section 12.6):

14.6 Lemma If s is a redex disjoint from r then its residual s^r is a redex of a^r. Moreover the two reductions commute, i.e. $(a^r)^{s^r} = (a^s)^{r^s}$. This result is shown as b in the diagram:

$$\begin{tikzcd} & a \arrow[dl, "r"'] \arrow[dr, "s"] & \\ a^r \arrow[dr, "s^r"'] & & a^s \arrow[dl, "r^s"] \\ & b & \end{tikzcd}$$

■

14.7 Reachability Let P be a subset of the nodes of an action a. The set of P*-reachable* nodes of a is the smallest set which includes P and all target-labelled (i.e. non-restricted) nodes, and such that if m is P-reachable and m carries n then n is P-reachable.

When $P = \{p\}$ we say p*-reachable*, and also talk of *reachable from* P or p, or (when P is empty) just *reachable*. It is clear that m is P-reachable iff it is reachable from some $p \in P$. We say an arc is P*-reachable* in a if it is carried by a P-reachable node. A node or arc which is not P-reachable is P*-unreachable*.

Intuitively, a reachable (i.e. $\emptyset$-reachable) arc of a is a particle which can eventually react with its environment without a performing any internal reactions. The number of external reactions needed to "reach" it is equal to the length of a path of (reachable) arcs leading to it from a target-labelled node.

We say a redex is P*-reachable* in a if its port is P-reachable.

There is a delicate interplay between reachable and unreachable redexes and nodes. If n is unreachable, then the firing of an *unreachable* redex can make n reachable; for the reduction may coalesce an unreachable node with a reachable one. In contrast, however, the firing of a *reachable* redex cannot improve reachability:

14.8 Lemma Let r be a P-reachable redex of a, and n be P-unreachable. Then n^r is P^r-unreachable in a^r.

Proof Prove the contrapositive; suppose n^r is P^r-reachable in a^r, and construct a path reaching n from P in a. ■

For our present purposes we also need the idea of a set of nodes which can only be reached from each other. We say that a set N of nodes in a is *isolated* if whenever $n \in N$ is P-reachable then $P \cap N \neq \emptyset$. In other words, every $n \in N$ is unreachable, and if it is p-reachable then $p \in N$.

As an example, for any P and a the set of P-unreachable nodes is isolated in a.

Let us say that r is an N*-redex* if its port is in N, and π is an N*-arc* if it is carried by a member of N. The importance of isolation lies in the following:

14.9 Lemma Let $a \searrow \ell$ where ℓ is a reachable action. Suppose that N is isolated in a, and that there is at least one N-arc. Then there is an N-redex r such that

$$a \overset{r}{\searrow} a^r \searrow \ell.$$

Proof Since N is isolated, the given N-arc is unreachable. So a is not reachable, and the reduction must be of length at least 1. The proof is by induction on the length of the reduction

$$a \overset{s}{\searrow} b \searrow \ell.$$

If the first redex s is an N-redex we are done. If not then it is P-reachable, where P is the complement of N in the node-set of a. But every node of N is P-unreachable in a, so by Lemma 14.8 every node of N^s is P^s-unreachable in b. In other words N^s is isolated in b; also there exists an N^s-arc. Hence by induction we have

$$a \overset{s}{\searrow} b \overset{t}{\searrow} c \searrow \ell,$$

where t is an N^s-redex in b. But s coalesces no nodes of N, so t is the residual r^s of an N-redex r in a. But r and s are disjoint redexes of a, so Lemma 14.8 yields, as required,

$$a \overset{r}{\searrow} a^r \overset{s^r}{\searrow} c \searrow \ell \,. \qquad \blacksquare$$

14.10 Preactions We need to establish some properties of arbitrary preactions. In each case it will be enough to verify them for the four basic forms of preaction given in Section 4.10.

Let φ be any preaction and a any action. For each node n of a we define its *image* a^φ in $\varphi(a)$ in the obvious way. Note that if φ is a precomposition then distinct nodes of a may have the same image. Similarly each arc π of a has an image π^φ; distinct arcs always have distinct images.

Any collection of arcs and nodes in a has an image in $\varphi(a)$. In particular

14.11 Lemma Let r be a redex in a. Then its image r^φ is a redex in $\varphi(a)$, and moreover $\varphi(a) \overset{r^\varphi}{\searrow} \varphi(a^r)$. $\blacksquare$

This property happens to hold even for *post*composition, not just for preactions, though we shall not need that fact. The next property is more crucial, because it holds for preactions but not for postcomposition:

14.12 Lemma Let N be isolated in a. Then N^φ is isolated in $\varphi(a)$. Further, if s is an N^φ-redex in $\varphi(a)$ then s is the image of an N-redex in a. $\blacksquare$

The proof of this is straightforward for the four basic preaction forms, and hence by an inductive argument for all preactions. Notice that it fails if φ is a postcomposition. Take for example $a = [\,\boldsymbol{\nu}x,\, \overline{x}\langle y\rangle\,]\langle x\rangle$, and $\varphi(-) = (-)\cdot b$ where $b = (x)[\,\overline{z}\langle x\rangle,\, x(w)\,]$. Now $\{x\}$ is isolated in a, but it is not isolated in

$$\varphi(a) \;=\; [\,\boldsymbol{\nu}x,\, \overline{x}\langle y\rangle,\, \overline{z}\langle x\rangle,\, x(w)\,]\;,$$

because z is free and carries x. Furthermore, $\varphi(a)$ has an x-redex which is the image of no redex in a. The postcomposition of b allows it to "gain access" to the isolated nodes (names) of a.

At last we are ready to prove the main theorem.

14.13 Proof of theorem 14.4 It is easy to verify that K^{RPIC} satisfies the first clause of Definition 7.1 (Incident-sets). For the second clause, we assume that $\varphi(a)\searrow\ell \in K^{\mathsf{RPIC}}$, and must find $k \in K^{\mathsf{RPIC}}$ for which $a\searrow k$ and $\varphi(k)\searrow\ell$.

We proceed by induction on the size (number of arcs) of a. If $a \in K^{\mathsf{RPIC}}$ we are done. If not then a has an unreachable arc π. Now let N be the unreachable nodes in a, so that π is an N-arc. Since N is isolated in a, from Lemma 14.12 we deduce that N^φ is isolated in $\varphi(a)$, and π^φ is an N^φ-arc.

By Lemma 14.9 there is an N^φ-redex s for which

$$\varphi(a) \overset{s}{\searrow} \varphi(a)^s \searrow \ell.$$

By Lemma 14.12, $s = r^\varphi$ where r is an N-redex in a. So let

$$a \overset{r}{\searrow} a';$$

then by Lemma 14.8 again $\varphi(a)^s = \varphi(a')$, so

$$\varphi(a') \searrow \ell.$$

Hence by induction there exists $k \in K^{\mathsf{RPIC}}$ such that

$$a' \searrow k \ \text{ and } \ \varphi(k) \searrow \ell,$$

and we are done since $a \overset{r}{\searrow} a'$. ■

15 Further work

The main contributions of this paper have been

- The definition of *action structures*, and the development of their intrinsic theory including the notions *effect* and *incident*;
- The uniform construction of a *process calculus* upon an arbitrary action structure, and a uniform theory of *bisimulation*;
- The exploration of particular action structures for the π-*calculus*, as an instance of the general theory.

In another paper [21] the notion of *action calculus*, a kind of syntactic action structure, has been defined. A natural further step, therefore, is to adapt the uniform construction of process calculi so that its result is indeed an action calculus; one must also extend the uniform theory of bisimilarity. As remarked at the start of Section 5 this is not a trivial step, but I believe it is possible; and if we claim that action structures provide a general framework then we should indeed carry out these constructions *within* it rather than *upon* it.

Already the examples in Section 3, as well as the π-calculus, show the action structure framework to be quite wide. It is therefore open to study at different levels. At one level we may explore the family of action structures for a particular model, e.g. Petri nets; the general theory may provide some insight for the particular model (e.g. a new kind of algebra of Patri nets). At a higher level, we may try to isolate some aspects of certain action structures to see whether they appear more widely. An example is the notion of *restriction*; it has already appeared in action structures both for π-calculus and for Petri nets, and it thus attains a more general significance. A more ambitious example (but linked with restriction) is the notion of *reflexion*. Reflexion arises naturally from the graphical presentation of π-nets (Section 13.4), yet is rather independent of the π-calculus; one may try to define the subclass of *reflexive action calculi*. It is also tempting to try to generalise the graphical presentation of PIC in Section 14; bearing in mind that graphical presentation of concurrent systems originated with Petri nets, which themselves form an action structure, one should surely try to isolate graphical notions which are common to a wide class of action structures.

Thus a main line of future work is to use action structures as a means of classification for models of concurrency; in this way, to echo the first sentence of this paper, we can hope to separate their ad hoc details from their essentials.

References

[1] Aceto, L. and Hennessy, M.C., *Towards action-refinement in process algebras*, To appear in Journal of Information and Computation.

[2] Austry, D. and Boudol, G., *Algèbre de processus et synchronisation*, Journal of Theor. Comp. Science, Vol 30, pp90–131, 1984.

[3] Baeten, J.C.M. and Weijland, W.P., **Process Algebra**, Cambridge University Press 1990.

[4] Barr, M. and Wells, C., **Toposes, Triples and Theories**, Springer Verlag, 1985.

[5] Bellin, G. and Scott, P.J. (with introduction by S. Abramsky), *On the π-calculus and linear logic*, Submitted for publication, 1992.

[6] Berry, G. and Boudol, G., *The chemical abstract machine*, Journal of Theoretical Computer Science, Vol 96, pp217–248, 1992.

[7] Best, E., Devillers, R. and Hall, J.G., *The box calculus: a new causal algebra with multi-label communication*, in **Advances in Petri Nets '92**, ed. G.Rozenberg, Lecture Notes in Computer Science, Springer Verlag, Vol.609, pp21–69, 1992.

[8] Brookes, S.D., Hoare, C.A.R. and Roscoe, A.W., *A theory of communicating sequential processes*, Journal of ACM, Vol 31, pp560–599, 1984.

[9] Cherkasova, L.A. and Kotov, V.E., *Descriptive and analytical process algebras*, in **Advances in Petri Nets '89**, ed. G Rozenberg, Lecture Notes in Computer Science, Springer Verlag, Vol 424, pp77–104, 1989.

[10] Freyd, P.J. and Kelly, G.M., *Categories of continuous functors, I*, Journal of Pure and Applied Algebra, Vol 2, pp169–191, 1972.

[11] Girard, J.-Y., *Linear logic*, J. Theoretical Computer Science, Vol 50, pp1–102, 1987.

[12] van Glabbeek, R. and Goltz, U., *Equivalence notions for concurrent systems and refinement of actions*, Proc. 4th Conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Springer Verlag, Vol 379, pp237–248, 1988.

[13] Gunter, C.A. and Scott, D.S., *Semantic domains*, in **Handbook of Theoretical Computer Science**, Vol A, pp633–674, Elsevier, 1990.

[14] Hoare, C.A.R., **Communicating Sequential Processes**, Prentice Hall, 1985.

[15] Lafont, Y., *Interaction nets*, Proc. 17th ACM Symposium on Principles of Programming Languages, 1990.

[16] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Journal of Theoretical Computer Science, Vol 96, pp73–155, 1992.

[17] Meseguer, J. and Montanari, U., *Petri nets are monoids*, Journal of Information and Computation, Vol 88, pp105–155, 1990.

[18] Milner, R., *Calculi for synchrony and asynchrony*, Journal of Theoretical Computer Science Vol 25, pp267–310, 1983.

[19] Milner, R., **Communication and Concurrency**, Prentice Hall, 1989.

[20] Milner, R., *The polyadic π-calculus: a tutorial*, in **Logic and Algebra of Specification**, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[21] Milner, R., *Action calculi, or concrete action structures*, Proc. MFCS Conference, Gdansk, Poland, Lecture Notes in Computer Science, Springer Verlag, Vol 711, pp105–121, 1993. (Also titled *Action calculi I: axioms and applications*, the first of a series on action calculi.)

[22] Milner, R., *Action calculi II: π-nets with boxes and replication*, Internal report, Laboratory for Foundations of Computer Science, Dept. of Computer Science, University of Edinburgh, 1993.

[23] Milner, R., *Action calculi III: higher-order*, submitted for publication. Internal report, Laboratory for Foundations of Computer Science, Dept. of Computer Science, University of Edinburgh, 1993.

[24] Milner, R., Parrow, J. and Walker D., *A calculus of mobile processes, I and II*, Journal of Information and Computation, Vol 100, pp1–40 and pp41–77, 1992.

[25] Nielsen, M., Plotkin, G.D. and Winskel, G., *Petri nets, event structures and domains*, Journal of Theoretical Computer Science, Vol 13, No 1, 1981.

[26] Nielsen, M., and Winskel, G., *Models for concurrency*, to appear in Volume IV of the **Handbook of Logic and the Foundations of Computer Science**, ed. D.Gabbay, Oxford University Press, 1993.

[27] Reisig, W., **Petri Nets**, EATCS Monographs on Theoretical Computer Science, ed. W.Brauer, G.Rozenberg, A.Salomaa, Springer Verlag, 1983.

[28] Winskel, G., *Petri nets, algebras, morphisms and compositionality*, Journal of Information and Computation, March 1987.