An Action Structure for Synchronous π-Calculus

Robin Milner

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

Abstract After a review of relevant notions, an action structure is presented for the π-calculus. This yields a version of π-calculus which is synchronous in the sense of MEIJE or SCCS, i.e. an arbitrary amount of computation may take place in a single transition. The main new technical result is the construction of an incident set for the action structure, which guarantees a congruential strong bisimilarity for the calculus. The incident set is characterized using a new form of graphical representation for actions.

1 Introduction

Action structures [7] have been proposed as a variety of algebra to underlie concrete models of concurrent computation and interaction. Over the last decade we have come to appreciate the difficulty of finding an abstract treatment of interactive computation, in the sense that Scott's domain theory is an abstract treatment of non-interactive computation. The progress with concrete models, notably Petri nets and process algebras, has led us to understand the wide range of choice to be made, as to what should or should not be abstracted away in such a treatment; consequently we tread more carefully in proposing abstractions.

Action structures aim not to make too much commitment as to the level of abstraction, but rather to unite the different concrete models. But there is danger in providing a general framework; to achieve generality can be to avoid understanding the specific nature of a topic. Action structures therefore make a commitment – not to a level of abstraction, but to the nature of the basic combinators for building concurrent systems. They take three combinators as basic: composition (of a certain kind), tensor product (representing parallel activity) and a form of parametrization. Added to these is a preorder representing the performance of actions; this performance can be represented either in full detail in a very concrete action structure, or in less detail in a more abstract one. An example was given in [7] of how to relate the two by a homomorphism of action structures.

In presenting a specific model –in particular the π-calculus [11]– in the framework of action structures, one therefore expects to add to the general theory of action structures what is specific to π-calculus.

One aspect of the general theory of [7] was a uniform way of building a process calculus from an action structure. Part of that theory is a treatment of bisimilarity, in terms of the notion of *incidents*. In this paper, a specific action structure for the π-calculus is presented, which –via the general theory– yields a synchronous version of the π-calculus, in exactly the way that MEIJE [1] or SCCS [9] is a synchronous version of calculi like CSP [3] or CCS [6].

The paper first reviews (in Sections 2 and 3) the theory of action structures; this makes the paper technically self-contained, though previous knowledge of action structures will certainly help. In Section 4 the action structure RPIC for the synchronous π-calculus is given with an illustration. (R stands for *reflexive*.) Section 5 describes the incident set for RPIC, which ensures a good notion of bisimilarity for processes of the synchronous π-calculus. Finally in Section 6 a graphical treatment of RPIC is outlined. A fuller treatment of all this work, with proofs, will be found in [10].

2 Action Structures, Effects and Incidents

In this section we recall from [7] the basic notions of action structures, up to and including *effects* and *incident sets*. We first define the notion *action structure* in category-theoretic terms, then elaborate it algebraically. Those unfamiliar with categories should find no difficulty if they focus upon the discussion following the definition.

2.1 Action structure: definition An action structure A is a strict monoidal category, with two extra items:

- a set X_A referred to as *names*, and for each $x \in X_A$ an endo-functor upon A known as an *abstractor*;
- a preorder $\searrow_A$ over each hom-set of A, called *reaction*, which is preserved by composition, monoidal product and abstraction, and for which the units are minimal. ■

Even more succinctly, an action structure is a preordered strict monoidal category with an indexed set of endo-functors.

2.2 Algebraic elaboration We shall work with the following algebraic characterization of action structures.

First, an action structure A has a monoid $(M_A, \otimes, \epsilon)$ of objects; these are the objects of the category and we shall call them *arities*. We shall use $k, l, m, n, \ldots$ to range over arities. The arities may be, for example, sequences of sorts (like INT, BOOL) under concatenation; in the special case when there is only one sort, 1, the monoid M_A is just the natural numbers under addition.

Second, for each pair m, n of arities A has a family $A_{m,n}$ of *actions*; they are the morphisms of A as a category. If a is a member of this set we write $a : m \to n$ and call m the *source arity* and n the *target arity* of a. We shall use a, b, c, d to range over actions.

Third, since A is a monoidal category with abstractors, there is a unit action $\mathbf{1}_m : m \to m$ for each arity m, and the operations of composition $\cdot$, tensor or monoidal product $\otimes$, and abstraction $\mathbf{ab}_x$ for each $x \in X_A$. They obey the following arity rules:

$$\mathbf{1}_m : m \to m \qquad \frac{a : k \to m \qquad b : m \to p}{a \cdot b : k \to p}$$

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

Note that composition is written forwards; we write $a \cdot b$ where the standard in category theory is to write $b \circ a$.

Note also that the operation of $\mathbf{ab}_x$ upon arities is not specified. In our intended applications there is usually an arity k_x associated with each names x, and then $\mathbf{ab}_x m$ is just $k_x \otimes m$. For example if arities are sort sequences we may have $k_x = \text{INT}$, which may mean that x is an integer variable; then for $a : \text{BOOL} \otimes \text{BOOL} \to \text{INT}$ we shall have $\mathbf{ab}_x a : \text{INT} \otimes \text{BOOL} \otimes \text{BOOL} \to \text{INT} \otimes \text{INT}$. In such cases $\mathbf{ab}_x$ is close to λ-abstraction; but in general we are not constrained to think of the set X_A as containing variables.

The status of A as a strict monoidal category with functorial abstractors is directly expressed by eight equational axioms, which we now give. Here (and later) we imagine arities to be ascribed in any way which respects the arity rules, and which gives the same arity to each side of an equation:

$$\begin{array}{ll}
a \cdot \mathbf{1} = a = \mathbf{1} \cdot a & a \cdot (b \cdot c) = (a \cdot b) \cdot c \\
a \otimes \mathbf{1}_\epsilon = a = \mathbf{1}_\epsilon \otimes a & a \otimes (b \otimes c) = (a \otimes b) \otimes c \\
\mathbf{1} \otimes \mathbf{1} = \mathbf{1} & (a \cdot b) \otimes (c \cdot d) = (a \otimes c) \cdot (b \otimes d) \\
\mathbf{ab}_x \mathbf{1} = \mathbf{1} & \mathbf{ab}_x (a \cdot b) = (\mathbf{ab}_x a) \cdot (\mathbf{ab}_x b).
\end{array}$$

Finally the reaction relation $\searrow_A$, which will often be written just $\searrow$, is a preorder on each $A_{m,n}$; so $a \searrow a'$ implies that a and a' have the same source and target arities. Reaction is also preserved by every operation; that is, $a \searrow a'$ implies $b \otimes a \searrow b \otimes a'$, $\mathbf{ab}_x a \searrow \mathbf{ab}_x a'$, etc. We stipulate that if $\mathbf{1} \searrow a$ then $a = \mathbf{1}$.

2.3 Intuition We think of tensor product ($\otimes$) as parallel composition. Reactions within a and within b can occur independently within $a \otimes b$. But there may be reactions $a \otimes b \searrow c$ which do not arise from a alone or from b alone; these represent communication or interaction between a and b. In particular, a and b may "use" the same name x, and the manner of use may constitute communication via x as a port.

Abstraction $\mathbf{ab}_x$ allows parametrization upon x. In particular, when x is a means of interaction as just described, then abstraction upon x allows one to vary the interface x, i.e. to vary the partners with whom communication via x may take place.

Composition $a \cdot b$ does not represent *sequential* composition. Given $a : m \to n$ and $b : n \to p$, the arity n indicates the extent to which b is parametric; to this extent, information may flow from a to b at any time during the continuing reaction within each of them. (It is sometimes better to think of an action as an *activity*, since it may enjoy an unbounded number of reactions.) This information can affect the further reaction within b; for example it may instantiate a parametric port (name) in b, as described in the previous paragraph, enabling interactions which were previously impossible.

The reader is invited to consider these discursive comments briefly at first, and later more thoroughly after considering the action structure for the π-calculus.

2.4 Effects of actions Consider briefly a transition in CCS [8], in which a value v is transmitted from one agent to another:

$$\overline{a}v.P \mid a(x).Q \xrightarrow{\tau} P \mid \{v/x\}Q.$$

We may think of the substitution $\{v/x\}$ here as the *effect* of the communication via the channel a; this effect influences Q in its further performance. We wish to capture the notion of effect in action structures, but in an abstract way which will allow effects other than substitutions. We shall think of an effect as a kind of degenerate action. Thus, if $a \searrow a' \cdot e$, where e is an effect, we imagine that no further reaction within a' or e, or between them, can withdraw the effect e. The most that can happen is that a' generates a further effect e', by the reaction $a' \searrow a'' \cdot e'$. We may think of the effects of a accumulating through the successive reactions

$$a \searrow a' \cdot e \searrow (a'' \cdot e') \cdot e = a'' \cdot (e' \cdot e),$$

where $e' \cdot e$ is also an effect. But though this accumulating effect cannot further influence the performance of a, each effect of a can influence b in the composition $a \cdot b$.

Let us now define effects precisely.

2.5 Inactivity and inertia: definition An action a is *inactive* if $a \searrow a'$ implies $a' = a$. An action a is *inert* if whenever $b \cdot a \searrow c$ then there exists b' such that $b \searrow b'$ and $c = b' \cdot a$. ■

Note that the units $\mathbf{1}_m$ are, trivially, inert in any action structure. It would not follow that they are inactive; but recall that we *required* them to be inactive in Definition 2.1. From this, it is an easy exercise to prove that *every* inert action is inactive.

We are going to demand that effects are inert actions. Intuitively, if e is inert the definition says that e cannot help any action b *precomposed* to it (i.e. in the composition $b \cdot e$) to react; so inertia captures something of the direction of information-flow.

We shall also place two further conditions upon effects, concerning their algebraic status. First, they must constitute a sub-actionstructure; the following definition is adapted from [7]:

2.6 Sub-actionstructure: definition A *sub-actionstructure* B of A consists of

- a subset X_B of the names X_A;
- a submonoid M_B of the arities M_A;
- for each pair of arities m, n a subset $B_{m,n}$ of the actions $A_{m,n}$, this family of subsets containing all the units and being closed under composition, product and abstraction;
- a reaction relation $\searrow_B$ which is the restriction of $\searrow_A$ to the actions in B. ■

Now for certain sub-actionstructures B we shall be interested in compositions $a \cdot b$ with $a \in A$ and $b \in B$. We call (a', b) a *decomposition of a for B* if $a = a' \cdot b$ and $b \in B$. We shall also refer to b as a *postcomponent* of a. We can preorder the decompositions of a for B by defining $(a_1, b_1) \leq (a_2, b_2)$ if some $b \in B$ makes the following diagram commute:

$$\begin{tikzcd} & {} \arrow[dr, "b_2"] \arrow[dd, "b"'] & \\ {} \arrow[ur, "a_2"] \arrow[dr, "a_1"'] & & {} \\ & {} \arrow[ur, "b_1"'] & \end{tikzcd}$$

It is easy to check that this is a preorder since B is a sub-actionstructure.

Now, thinking of the B potentially as the effects which actions may have, we want to find a condition which allows us to speak about *the* effect of an action a, meaning in some sense the accumulation of all postcomponents of a which lie in B. The weakest condition on B which makes sense of this is that B is a postcomponent of A (note that we shall now use this term for sub-actionstructures as well as for actions), according to the following:

2.7 Postcomponent: definition A sub-actionstructure B of A is a *postcomponent* of A if for every $a \in A$ the decompositions of a for B form a directed set under $\leq$. ■

$$\begin{tikzcd} & & {} \arrow[dr, "b_1"] & \\ {} \arrow[urr, "a_1"] \arrow[r, "a'"] \arrow[drr, "a_2"'] & {} \arrow[ur, "b'_1"] \arrow[dr, "b'_2"'] & & {} \\ & & {} \arrow[ur, "b_2"'] & \end{tikzcd}$$

The picture shows what is meant; given two decompositions (a_1, b_1) and (a_2, b_2) of the same action, there must exist $a' \in A$ and $b'_1, b'_2 \in B$ to make the diagram commute.

We are now ready to define the notion of effect.

2.8 Effect structure: definition A sub-actionstructure E of A is an *effect structure for A* if E is a postcomponent whose members are all inert. ■

To complete the groundwork for building process calculi upon arbitrary action structures, we need to develop the notion of *incident*, upon which the uniform treatment of process bisimilarity will rest. Conveniently, we can link incidents to effects via *preactions*, which we now define.

2.9 Preactions and strong inertia A *preaction* is a unary operation $\varphi(-)$ over A gained by composing operations of the form $c \cdot (-)$, $c \otimes (-)$, $(-) \otimes c$ and $\mathbf{ab}_x(-)$ where c may be any action. Thus $\mathbf{ab}_x(b \otimes (c \cdot (-)))$ is a preaction, but $b \otimes ((-) \cdot c)$ is not since it uses postcomposition, $(-) \cdot c$.

Recall that an effect e is required to be inert; that is, e cannot "take part" in any reaction when placed in a precomposition context $c \cdot (-)$. The first important property of preactions is that, as it turns out, an effect is incapable of partaking in a reaction in *any* preaction context φ (not just precomposition); that is, whenever $\varphi(e) \searrow b$ then there is some preaction φ' such that $\varphi \searrow \varphi'$ and $\varphi'(e) = b$. This property was called *strong inertia* in [7].

Preactions also play an important part in the following:

2.10 Incident sets: definition A subset K of A is an *incident set* if it satisfies the following two conditions, for every $a \in A$:

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

Due to the first clause, the notion of incident set is relative to a given effect structure. An incident set need not be a sub-actionstructure of A. Note that there may be many incident sets relative to a given E; in particular, both A and the empty set are always incident sets. However, it was shown in [7] that under a mild condition there is always a *least* incident set containing any given set of actions (e.g. the effects).

The rôle played by an incident set will be as follows. When we have defined processes over an arbitrary action structure, we shall define *transitions* $\xrightarrow{a}$ labelled by actions; then in terms of these transitions and the notion of effect we define bisimilarity over processes. If we use the transitions $\xrightarrow{a}$ for *all* actions a then our bisimilarity relation will be a congruence, but a very fine congruence. If we restrict a to range over an arbitrary subset of A, then we get a coarser equivalence which in general will not be a congruence. But if we restrict a to range over an incident set, then a congruence is guaranteed. Furthermore, a particular congruence is distinguished by taking the least incident set containing the effects E.

3 Process Calculi

3.1 Grammar and structural congruence In a previous paper [7] a uniform way of extending an arbitrary action structure A to a process algebra was given. Letting a range over A, the grammar of *processes* (which we shall denote by $P, Q, \ldots$) is[1]

$$\begin{array}{rcl} P & ::= & a \cdot P \mid P \otimes P \mid (x)P \\ & & \bullet P \mid \mathbf{0} \mid P + P \mid \partial P \mid !P \end{array}$$

subject to the following arity rules, which ascribe arities $m \in M_A$ to processes:

$$\frac{a : m \rightarrow n \qquad P : n}{a \cdot P : m} \qquad \frac{P : m \qquad Q : n}{P \otimes Q : m \otimes n} \qquad \frac{P : m}{(x)P : \mathbf{ab}_x m}$$

[1] The notation $(x)P$ is used here, where $\mathbf{ab}_x P$ was used in the previous paper [7]. It seems better to reserve the latter for the functorial operation.

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

Of these process constructions, the first three lift the action constructions to the level of processes – but only admitting composition between an action and a process, not between two processes. We impose a *structural congruence* upon the grammar, lifting the action structure algebra to processes as far as possible. Structural congruence is the smallest congruence relation $\equiv$ satisfying these four laws:

$$\begin{array}{rcl} P & \equiv & \mathbf{1} \cdot P \\ a \cdot (b \cdot P) & \equiv & (a \cdot b) \cdot P \\ (a \cdot P) \otimes (b \cdot Q) & \equiv & (a \otimes b) \cdot (P \otimes Q) \\ (x)(a \cdot P) & \equiv & (\mathbf{ab}_x a) \cdot (x)P \ . \end{array}$$

From now on we shall elide the composition symbol $\cdot$, writing ab for $a \cdot b$ and aP for $a \cdot P$. We also give (elided) composition highest binding power and all other binary operators lowest binding power, so that $ab \otimes cd$ means $(ab) \otimes (cd)$, $(x)aP$ means $(x)(aP)$, $\mathbf{ab}_x c \otimes d$ means $(\mathbf{ab}_x c) \otimes d$ and $\boldsymbol{\partial} P + Q$ means $(\boldsymbol{\partial} P) + Q$.

3.2 Control operators and dynamics The last five process constructions are called *control terms*, constructed by the *control operators* $\bullet, \mathbf{0}, +, \boldsymbol{\partial}$ and $!$. They are called respectively *committal*, *zero*, *sum*, *delay* and *replication*. Their meaning is give not by equations but by reduction rules; these provide the dynamics of processes.

We say that an occurrence in P of an action a or process term Q is *ready* if it is not inside any control term. A *context* $\mathcal{C}$ is a process term with a single hole, which is denoted by [ ], to be filled by an action or 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.

We now define a one-step reduction relation reduction relation $\searrow^1$ over processes; it is the union of (one-step) *action reduction* $\searrow^1_{\mathbf{a}}$ and (one-step) *control reduction* $\searrow^1_{\mathbf{c}}$. There is a single rule for action reduction:

$$\text{if } a \searrow a' \text{ and } \mathcal{C} \text{ is ready then } \mathcal{C}[a] \searrow^1_{\mathbf{a}} \mathcal{C}[a'].$$

For control reduction there are two rules each for sum, delay and replication and two rules for committal:

$$\begin{array}{lll} P + Q \searrow^1_{\mathbf{c}} P & \qquad \boldsymbol{\partial} P \searrow^1_{\mathbf{c}} P & \qquad !P \searrow^1_{\mathbf{c}} P \otimes\, !P \\ P + Q \searrow^1_{\mathbf{c}} Q & \qquad \boldsymbol{\partial} P \searrow^1_{\mathbf{c}} \bullet \boldsymbol{\partial} P & \qquad !P \searrow^1_{\mathbf{c}} \bullet\, !P \end{array}$$

$$\begin{array}{rcl} \bullet P \otimes \bullet Q & \searrow^1_{\mathbf{c}} & \bullet (P \otimes Q) \\ (x) \bullet P & \searrow^1_{\mathbf{c}} & \bullet (x)P \end{array}$$

together with the context rule

$$\text{if } P \searrow^1_{\mathbf{c}} P' \text{ and } \mathcal{C} \text{ is ready then } \mathcal{C}[P] \searrow^1_{\mathbf{c}} \mathcal{C}[P'].$$

We also define $\searrow$, $\underset{a}{\searrow}$ and $\underset{c}{\searrow}$ as the transitive reflexive closures of $\searrow^1$, $\underset{a}{\searrow}^1$ and $\underset{c}{\searrow}^1$. We take all these relations to be closed under structural congruence.

The purpose of these rules is to recover, in a general framework, the familiar notion of transition in existing process calculi. We shall shortly define labelled transitions $P \xrightarrow{a} P'$ in terms of reductions of the special form

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

We shall call such a reduction a *commitment*; it reduces P to the composition of an action with a committal. Consider for example the following transition in SCCS:

$$a \bullet P \times \boldsymbol{\partial}\overline{a} \bullet Q \xrightarrow{\mathbf{1}} P \times Q;$$

in this framework it will be inferred as follows:

$$\begin{array}{rcll} a \bullet P \otimes \boldsymbol{\partial}\overline{a} \bullet Q & \underset{c}{\searrow}^1 & a \bullet P \otimes \overline{a} \bullet Q & \text{(first } \boldsymbol{\partial} \text{ rule)} \\ & \equiv & (a \otimes \overline{a})(\bullet P \otimes \bullet Q) & \\ & \underset{c}{\searrow}^1 & (a \otimes \overline{a}) \bullet (P \otimes Q) & \text{(first committal rule)} \\ & \underset{c}{\searrow}^1 & \mathbf{1} \bullet (P \otimes Q) & \text{(action reduction rule).} \end{array}$$

Another commitment can also be inferred for this process, namely

$$a \bullet P \otimes \boldsymbol{\partial}\overline{a} \bullet Q \quad \searrow \quad a \bullet (P \otimes \boldsymbol{\partial}\overline{a} \bullet Q).$$

We are now ready to define labelled transitions over processes. As we said above, they will be defined in terms of commitments, $P \searrow a \bullet P'$. We wish to allow any effect of a in such a commitment to act upon P', which is achieved as follows:

3.3 Transition: definition Given an action structure A, the *transition relations* $\{\xrightarrow{a} \mid a \in A\}$ are defined to be the smallest binary relations on processes over A which satisfy the rule

$$\frac{P \searrow ae \bullet P'}{P \xrightarrow{a} eP'}.$$

■

These relations are the basis for a uniform definition of bisimulation.

3.4 Bisimulation: definition Let K be any set of actions in A. A binary relation $\mathcal{S}$ over processes is a K-*simulation* if, for all $(P, Q) \in \mathcal{S}$ and $k \in K$, if $P \xrightarrow{k} P'$ then there exist $k' \in K$,$e \in E$ 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 the pair (P, Q) is in some K-simulation then we say that Q K-*simulates* P and write $P \lesssim_K Q$. If (P, Q) is in some K-bisimulation then we say that P and Q are K-*bisimilar* and write $P \sim_K Q$. ■

It is easy to check that K-similarity is a preorder, and K-bisimilarity is an equivalence, provided K satisfies the condition that $a \in K$ iff $ae \in K$, for all $a \in A$ and $e \in E$ such that the composition ae is defined.

We naturally seek general conditions upon K which ensure that the relations will be *congruential*, i.e. preserved by all the process constructions. The following was proved in [7]:

Theorem If K is an incident set then $\lesssim_K$ is a precongruence and $\sim_K$ a congruence over processes. ∎

4 An Action Structure for the π-Calculus

A family of action structures for the π-calculus is discussed in detail in [10]. Space allows only one of these to be presented here. We choose the one which is perhaps the most striking; this provides a process calculus which bears the same relation to the standard π-calculus [11] as MEIJE [2] or SCCS [9] bears to CSP [3] or CCS [6]. It has the mobility of π-calculus, and also the power to perform an arbitrary amount of computation within a single action. The action structure also has a pleasant graphical presentation.

4.1 The action structure RPIC In [7], Section 3, the action structure SCCS is defined. The body of each action is a multiset over $\Pi \cup \overline{\Pi}$, where Π is a set known as the (positive) particles, and $\overline{\Pi} = \{\overline{\pi} \mid \pi \in \Pi\}$ is a set known as the negative particles, disjoint from Π. Reaction consists simply in cancelling a complementary pair $\{\pi, \overline{\pi}\}$. SCCS also allows some particles in an action to be restricted.

To get RPIC 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 RPIC are now given by

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

The first two kinds are *input* and *output* (collectively *i/o*) particles, of positive and negative sign respectively. In the input particle $x(\vec{y})$ the vector $\vec{y}$ consists of distinct names, which are binding occurrences. The name in a restriction particle is also binding.

The general form of an action $a : m \to n$ of RPIC is [2]

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

where

- $\vec{x}$ is an m-vector of distinct names, the *imported* names;

- M is a multiset $[\,\pi_1 \ \cdots \ \pi_k\,]$ of particles, the *body*;

[2]The arities in RPIC are the integers under addition. A natural refinement is to take them to be sort-sequences, as in [8].

- $\vec{y}$ is an n-vector of names, not necessarily distinct, the *exported* names.

The imported names are bound. There must be at most one binding occurrence of any name (including binding occurrences in particles of M). The scope of each bound name is the entire action, not just the part to the right of the binding occurrence. We tacitly assume in what follows that by suitable alpha-conversion (change of bound names) a name bound in an action is not used elsewhere in any actions under discussion.

As examples, we immediately define five important kinds of action.

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

We shall allow ourselves to omit any empty component of an action; so we may write $[\,\tau\,]$ for $(\,)[\,\tau\,]\langle\,\rangle$, and $(\vec{y})[\,\overline{x}\langle\vec{y}\rangle\,]$ 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.

4.2 Operations The action structure operations for RPIC are defined just as for the action structure SCCS in [7]; in fact SCCS turns out to be a sub-actionstructure of RPIC. First, the identities are given by

$$\mathbf{1}_m \stackrel{\text{def}}{=} (\vec{x})\langle\vec{x}\rangle \qquad (|\vec{x}| = m).$$

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 $\vec{v}$. If $a = (\vec{u})M\langle\vec{v}\rangle$ and $b = (\vec{x})N\langle\vec{y}\rangle$ then

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

Recall our convention that a bound name differs from all names used outside its scope. Thus in $a \otimes b$ no particle in M or N can bind a name which occurs in the other; the tensor product is simply the juxtaposition of the two actions. But in $a \cdot b$ the substitution σ may replace some names free in N by names bound in M.

Besides the action structure operations we define the following, again assuming $a = (\vec{u})M\langle\vec{v}\rangle$:

$$
\begin{array}{rcll}
(x)a & \stackrel{\text{def}}{=} & (x\vec{u})\, M\, \langle\vec{v}\rangle & \\
(\boldsymbol{\nu} x)a & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} \otimes \mathbf{1}) \cdot (x)a & (\,= (\vec{u})\,[\boldsymbol{\nu} x] \cup M\, \langle\vec{v}\rangle\,).
\end{array}
$$

4.3 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 said to be a *redex* of a, with *port* u. Roughly speaking, to reduce a redex we erase its particles and apply the substitution $\{\vec{v}/\vec{w}\}$; this corresponds to the rule of communication in the π-calculus [11]. But there is an important difference, which is the reason that we call RPIC *reflexive*; in contrast with the π-calculus we shall allow members of $\vec{v}$ to be among the bound names $\vec{w}$. The simplest case of this is the redex $\overline{u}\langle v\rangle\ u(v)$, where v in the first particle is bound by the second. At first sight it seems nonsense to generate the self-substitution $\{v/v\}$; a name surely cannot denote itself! Well, we argue that this is a way to ensure that it never denotes *anything else*; and this is precisely what restriction is supposed to ensure. We therefore dictate that

$$\overline{u}\langle v\rangle\ u(v)\ \searrow^1\ \boldsymbol{\nu} v.$$

More generally we define the one-step reduction relation $\searrow^1$ as follows: if a contains the redex $\overline{u}\langle\vec{v}\rangle\ u(\vec{w})$ then

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

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. 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 if we reorder the substitution elements we get

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

which is the same up to alpha-conversion.

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

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

■

Note that the port of a redex (in the example, first u then x) may be free or bound.

4.4 Effects What should be the effect structure for RPIC? There is no single answer; to simplify the decision, we shall consider only actions whose body contains no i/o 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 Paragraph 4.5. But the class contains some undesirables; one such is $\boldsymbol{\omega}$, which discards a name. It is undesirable because discarding can be made into a postcomponent of *any* action; for example if $a : 1 \rightarrow 1$ 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 RPIC consist of all actions

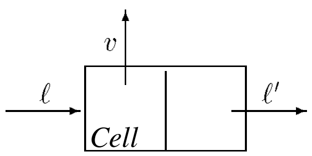
$$(\vec{x})[\,\boldsymbol{\nu}\vec{u}\,]\langle\vec{y}\rangle, \qquad \vec{x} \cup \vec{u} \subseteq \vec{y}.$$

It is routine to verify that E is indeed an effect structure.

4.5 Illustration Let us now illustrate the power of composite action in RPIC, when extended by process structure as described in Section 3. We shall use one further notation; recalling that $\boldsymbol{\nu}$ is an action, we define restriction upon processes P of arity 0 by

$$\begin{array}{rcl} (\boldsymbol{\nu} x)P & \stackrel{\text{def}}{=} & \boldsymbol{\nu} \cdot (x)P \\ (\boldsymbol{\nu}\vec{x})P & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} x_1)(\boldsymbol{\nu} x_2)\cdots P \end{array}$$

We shall use an encoding of data structures as processes, close to that described in [8]. Specifically, we consider list structures. 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 $[\,\ell(cn)\,\overline{c}\langle v\ell'\rangle\,] : 0 \rightarrow 0$.



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

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

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

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

For now, let us understand a *value* 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". (Note that in this sense *Cell* and *Nil* are not values. But there are variations on this theme, as was indicated in [8].) A located value is naturally written as $\langle v\rangle Val$ (i.e. $\langle v\rangle \cdot Val$), because $\langle v\rangle$ is indeed an action. So what is the list containing k values $Val_1, \ldots, Val_k$? It is itself a value $L : 1 \to 0$, given by

$$\begin{aligned} L \quad = \quad & (\ell_0)\,!(\boldsymbol{\nu} v_1\ell_1 \cdots v_k\ell_k)\Big(\,\langle \ell_0 v_1 \ell_1\rangle Cell \otimes \cdots \otimes \langle \ell_{k-1} v_k \ell_k\rangle Cell \otimes \langle \ell_k\rangle Nil \\ & \qquad\qquad\qquad\qquad\qquad \otimes \langle v_1\rangle Val_1 \otimes \cdots \otimes \langle v_k\rangle Val_k\,\Big). \end{aligned}$$

We next define the single actions *head* and *tail*, which get hold of the first element and the remainder of an arbitrary list value. They are given by

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

Now 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 $\langle l_0\rangle head$ to P; then in a single transition this action gains access to the head of L, as follows:

$$\langle \ell_0\rangle head \,\bullet\, P \otimes \langle \ell_0\rangle L \;\xrightarrow{\;b\;}\; (\boldsymbol{\nu} v\cdots)\Big(\,\langle v\rangle P \otimes \langle v\rangle Val_1 \otimes \langle \ell_0\rangle L\;\cdots\,\Big).$$

We see that P has indeed "received" v, the location of L's first member Val_1. Note that, due to replication, the located list is still available for further access. The dots $\cdots$ represent material which is irrelevant for present purposes. The action b will normally be an incident; 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 bewteen P and the value Val_1.

Each step down the list L need not involve a separate transition. The following single transition provides access to the fourteenth member of L, where $tail^n$ means the composition of n *tail*s:

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

This example prompts a natural question: Why should it be necessary to distinguish processes from actions? Why should we not add the control operations $\bullet$, $\mathbf{0}$, $+$, $\boldsymbol{\partial}$ and $\mathbf{!}$ 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. A 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.

5 An Incident Set for RPIC

5.1 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 [7] 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.

Now RPIC 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:

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

Indeed it can be shown that *any* non-empty incident set for RPIC must contain active members. We now proceed to exhibit an interesting and natural incident set for RPIC.

5.2 Reachability We now seek to understand how one reaction may (or may not) preclude another; this helps us to decide which actions should be incidents.

Suppose that an action 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 alone, 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 2.10. A further consideration is that an incident should contain no particle which cannot eventually take part in some reaction.

The definitions which follow should be understood with these ideas in mind. Recall that we assume every bound name (in the actions under discussion) to be distinct from every other bound name, and from all names occurring free.

Definition The set of names *reachable* in a given action $a \in \mathsf{RPIC}$ 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. ■

The main result of this paper is then as follows:

5.3 Theorem The set of reachable actions is an incident set in RPIC. ■

The proof of this result appears in [10], and is not trivial; I shall not give it here. But it has one attractive aspect; it seems best conducted by using graph-theoretic methods. Indeed the notion of reachability suggests a graphical treatment; and it turns out that there is a pleasant graphical presentation of RPIC, which is briefly outlined in the next section.

I also conjecture, but have not proved, that the reachable actions are the smallest incident set for RPIC containing the effects, provided only that we omit any action containing a *nullary* redex – one of the form $\overline{x}\langle\;\rangle\; x(\;)$ carrying no names.

It should be emphasized that there is no guarantee of a decent incident set, and hence of a tractable process calculus. The reflexive π-calculus arose from my failed attempt to find an incident set for PIC, the non-reflexive version of RPIC. In PIC the body of an action is not a multiset but a *partial sequence* $[\,\pi_1\;\cdots\;\pi_k\,]$ of particles, where the scope of each bound name extends to the *right* of its binding occurrence (not to the whole action), and any commutation $\pi\pi' = \pi'\pi$ of adjacent particles is allowed provided it does not render a bound name free or vice versa. Moreover a redex is a partial sequence $\overline{x}\langle\vec{u}\rangle\; x(\vec{v})$, where the names $\vec{u}$ cannot be among the bound names $\vec{v}$. PIC seemed the obvious action structure for the π-calculus; but it turns out that the reachable actions are *not* an incident set for it; nor do I know of any interesting incident set. (I am grateful to David Turner for helping me to find this anomaly.) Surprisingly enough, if we take the *monadic* sub-actionstructure of PIC, then again the reachable actions are an incident set, though the proof is somewhat less easy than for RPIC.

A conclusion to be drawn from this is that the existence of a decent incident set is a non-trivial criterion by which to judge an action structure.

To finish this section, we return briefly to our illustration.

5.4 Illustration revisited Consider again the list-processing illustration in Paragraph 4.5. Let us make a slight change, but one which makes a crucial difference to the incidents involved. We now suppose that the process $\langle \ell_0 \rangle$*head*$\bullet P$ is to have exclusive access via ℓ_0 to the list L, so we restrict ℓ_0 and consider the behaviour of $(\boldsymbol{\nu}\ell_0)(\ \langle \ell_0 \rangle \mathit{head} \bullet P \otimes \langle \ell_0 \rangle L\ )$. It has a large number of possible commitments

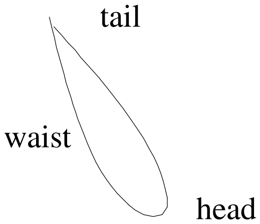
$$(\boldsymbol{\nu}\ell_0)\Big(\ \langle \ell_0 \rangle \mathit{head} \bullet P \otimes \langle \ell_0 \rangle L\ \Big) \searrow a \bullet Q;$$

but in almost all of them the action a will not be an incident. They correspond either to "erroneous" commitments by the component cells of L, or else to actions which have not yet performed their purely internal reactions. Indeed, it turns out that there is *only one* commitment in which a is an incident; it correspond to correct and complete performance of the access to the list's first member. The same is true for any other access, e.g. to the fourteenth member. So it appears that by confining attention to transitions labelled by incidents, one indeed models the intended computations and no others. Of course, we cannot claim to have stated this claim rigorously, much less to have proved it. That is a challenge for further analysis.

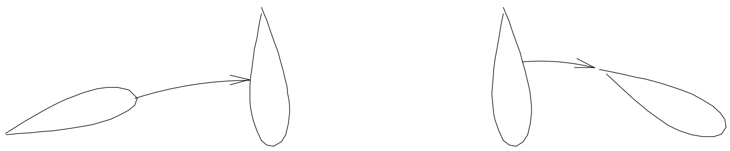
6 Graphical presentation

The action structure RPIC has been presented in a very syntactic way. This helps formal manipulation, but some essential aspects are still hidden. Here we indicate briefly that both the statics and the dynamics of RPIC correspond to simple graphical ideas. Fuller details are given in [10]. This graphical form may be called a *geometry of interaction*; it is similar in spirit to the work of Lafont [5], and of Girard [4] on proof-nets in linear logic.

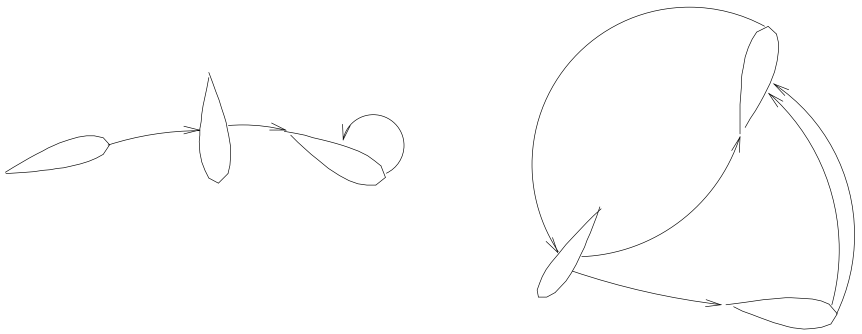
6.1 Bare π-nets We begin with a simple kind of graph called a *bare π-net*. It has *nodes* and *directed arcs*. Each node is drawn as a pointed oval with a *head*, a *waist* and a *tail* as shown.



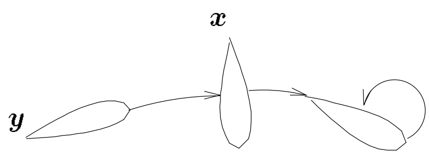
Each directed arc is either a *source* arc or a *target* arc. We shall only illustrate *monadic* arcs; the polyadic arcs needed to represent the polyadic π-calculus are a straightforward generalization. A monadic source arc is drawn from the head of a node (the *source* of the arc) to the waist of a node (the *port*); a monadic target arc is drawn from the waist of a node (the *port*) to the tail of a node (the *target*).



The only constraint in forming π-nets from such nodes and arcs is that each node is the target of at most one arc. Here are two bare monadic π-nets:



6.2 Representing RPIC **by π-nets** Every bare π-net represents an action of RPIC with no free names. We now describe those π-nets which represent actions of arity $0 \to 0$. Consider the action $a = [\,\overline{x}\langle y\rangle\ x(z)\ \overline{z}\langle z\rangle\,]$; since x occurs free in a, there is a node in the π-net for a which is not a target but bears the name x; similarly for y. Every node in the net corresponds to a name occurring (free or bound) in a, but those representing bound names bear no labels. Every output particle $\overline{u}\langle v\rangle$ is represented by a source arc whose port is (the node representing) u and whose source is the node v. Similarly for every input particle there is a target arc. In fact, the π-net for a is just the first of the above nets but with two labelled nodes, thus:

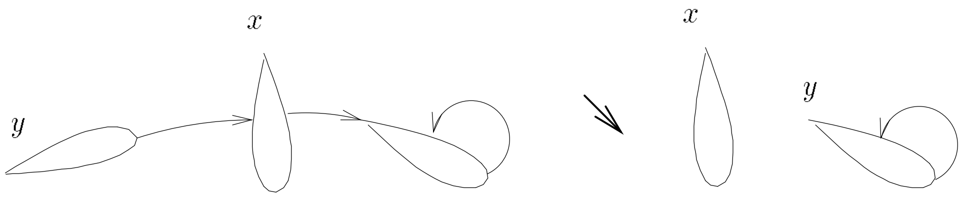


$[\,\overline{\boldsymbol{x}}\langle \boldsymbol{y}\rangle\ \boldsymbol{x}(\boldsymbol{z})\ \overline{\boldsymbol{z}}\langle \boldsymbol{z}\rangle\,]$

A node which is not a target and bears no label represents a restricted name. So the above net *without* labels represents the action $[\,\boldsymbol{\nu} xy\ \overline{x}\langle y\rangle\ x(z)\ \overline{z}\langle z\rangle\,]$.

To represent actions of arbitrary arity, we only need to do a little more labelling; we shall omit the details. Then the operations of composition, product and abstraction may be defined as simple combinations of π-nets, involving the coalescing of certain nodes.

It remains to outline reaction. It should now be clear that a redex consists of a source arc and a target arc with the same port. Thus the two arcs carried by the port x in the above net are a redex; here is its reduction:



$$[\,\overline{x}\langle y\rangle\; x(z)\;\overline{z}\langle z\rangle\,] \searrow [\,\overline{y}\langle y\rangle\,]$$

Reducing a redex consists in removing the two arcs and coalescing the source and target nodes (leaving the port node unchanged) so that a single node now bears the remaining arcs and label(s) of both. Note that the node x has become detached in the example shown; it carries no further arcs. Such nodes can be ignored.

We shall not take this representation any further. We hope to have suggested that notions such as reachability should be easy to handle in terms of the graphs. In proving that reachable actions form an incident set, the benefit of the graphical form is to avoid the distraction of syntax.

References

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

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

[3] 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.

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

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

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

[7] Milner, R., *Action structures*, Research Report LFCS–92–249, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992.

[8] Milner, R., *The polyadic π-calculus: a tutorial*, Research Report LFCS–91–180, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1991. (To appear in the Proceedings of the International

Summer School on Logic and Algebra of Specification, Marktoberdorf, August 1991.)

[9] Milner, R., *Calculi for synchrony and asynchrony*, Journal of Theoretical Computer Science, Vol.25, pp 267–310, 1983.

[10] Milner, R., *Action structures for the π-calculus*, Forthcoming research Report, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1993.

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