Action Calculi I: Axioms and Applications

Robin Milner
Edinburgh, June 1993

Abstract Action structures have previously been proposed as an algebra for both the syntax and the semantics of interactive computation. Here a class of concrete action structures called action calculi is identified, which can serve as a non-linear syntax for a wide variety of models of interactive behaviour. They generalise a previously defined action structure PIC for the π-calculus. One action calculus differs from another only in its generators, called *controls*.

Several extensions to PIC are given as action calculi, giving essentially the same power as the π-calculus. An action calculus is also outlined for PT nets – a class of Petri nets – parametrized upon their places and transitions.

Finally, action calculi are characterized as the free algebras in a sub-variety of action structures, namely those which satisfy certain additional axioms. [1]

1 Introduction

Action structures [7] have been proposed as an algebra for both the syntax and the semantics of interactive computation. In particular, the cited paper gave an example expressing the meaning of a simple language (for evaluating expressions) as a homomorphism between two action structures, one expressing the evaluation mechanism and the other containing the denoted entities. One hopes to extend this treatment to formalisms which express interaction; then the framework will apply not only to programming languages but to the wider class of communicating systems. Formidable difficulties remain in determining what the denoted entities should be in this case; one further hopes that action structures will assist in tackling this problem.

The present paper addresses a question crudely expressed thus: When is an action structure syntactical? Various authors have found it helpful to use formalism with richer structure than a free term algebra, when defining an evaluation mechanism or a communicational model for concurrent systems. Following ideas of Banâtre and Métayer [2], Berry and Boudol based their Chemical Abstract Machine [1] on the notion of a *multiset*. Prompted by them, the present author imposed a structural congruence upon the π-calculus as part of the formal language, not of the semantics [6]. Again, Meseguer and Montanari [5] have revealed the monoidal structure inherent in Petri nets. Indeed, part of the very success of Petri's notion of a net [11] is that its syntax is graphical, not linear.

By imposing an enriched monoidal structure upon action structures, we are already committed to this path; in fact action structures go further and include laws of abstraction (parametrization) over names. But the action structure axioms alone do not determine

[1] This is essentially the paper presented at MFCS'93, in Gdansk, under the title "Action Calculi, or Syntactic Action Structures". Several short notes are being written which develop the notion; it seems useful to make this evident in their titles, "Action Calculi I: . . . " etc., starting with this one.

a syntactic subclass of action structures. Here we present such a class, which we call *action calculi*.

By adding six so-called *concrete* axioms to the basic ones, we arrive at action calculi as term algebras quotiented by the congruence which the axioms induce. These term algebras have an appealing concrete presentation. One action calculus differs from another only in its generators and their dynamics; the generators are best seen as *control operators*. It turns out that the action structure PIC for the synchronous π-calculus [9] is an action calculus, and has just three generators $\boldsymbol{\nu}$, **in** and **out** (restriction, input and output), while an action calculus for Petri's place-transition nets is generated by $\boldsymbol{\nu}$, **pre**, **post** and **mark** (restriction, pre-condition, post-condition and marking). Thus action calculi begin to achieve one of the main purposes of action structures – to unite different models of concurrency in a common setting.

The paper is organised as follows. Section 2 reviews the basic definition of action structures. Section 3 reviews the action structure PIC, which serves as an exemplar for the general definition of action calculus presented in Section 4. Section 5 shows how PIC can be extended to richer action calculi containing guarding, choice and replication, by adding further generators; it also outlines the treatment of Petri nets in an action calculus called NETC. Section 6 is technical; it gives the concrete axioms, substantiates the claim that each action calculus is indeed a quotient of a term algebra, and sketches the category of *concrete action structures*, in which action calculi are the initial objects. Section 7 briefly identifies some future lines of investigation.

2 Action structures reviewed

In this section we recall from [7] the basic notion of an action structure. We first give it in category-theoretic terms, then elaborate it algebraically. Those unfamiliar with categories should find no difficulty if they focus upon the algebra.

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. An arity may be, for example, a sequence 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 and n the *source* and *target* arities of a; we even abuse terminology by calling $m \to n$ just the 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 $\mathbf{id}_m : m \to m$ 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*:

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

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

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. But throughout this paper we shall assume an arity k_x associated with each name 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 $\mathbf{ab}_x a : \text{INT} \otimes \text{BOOL} \otimes \text{BOOL} \to \text{INT} \otimes \text{INT}$.

The status of A as a strict monoidal category with functorial abstractors is directly expressed by eight basic 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:

BASIC AXIOMS	
$a \cdot \mathbf{id} = a = \mathbf{id} \cdot a$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
$a \otimes \mathbf{id}_\epsilon = a = \mathbf{id}_\epsilon \otimes a$	$a \otimes (b \otimes c) = (a \otimes b) \otimes c$
$\mathbf{id} \otimes \mathbf{id} = \mathbf{id}$	$(a \cdot b) \otimes (c \cdot d) = (a \otimes c) \cdot (b \otimes d)$
$\mathbf{ab}_x \mathbf{id} = \mathbf{id}$	$\mathbf{ab}_x(a \cdot b) = (\mathbf{ab}_x a) \cdot (\mathbf{ab}_x b)$.

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 $\mathbf{id} \searrow a$ then $a = \mathbf{id}$.

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 which 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 or between them. (It is sometimes better to think of an action as an *activity*, since it may enjoy an unbounded number of reactions.)

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.

3 An action structure for the π-calculus

In this section we define an action structure PIC for the π-calculus. It is studied more fully, with some alternatives, in [9]; here we shall use it as an exemplar and motivation for our later definition of action calculi.

3.1 Particles PIC formalises two principal features of the π-calculus; the passage of names through ports which are also names, and the localisation of names by restriction. Thus the elementary constituents of an action in PIC are the *particles* π, 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* particles. In the input particle $x(\vec{y})$ the vector $\vec{y}$ consists of distinct names, which are binding occurrences. The third kind is a *restriction particle*, and its name occurrence is also binding.

3.2 Actions The essential part of an *action* of PIC is just a collection of particles. It would be a multiset, were it not for the binding discipline. 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. Thus the first occurrence of y binds the second. Adopting the convention that the scope of a binding extends to the right, 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, here are five important kinds of action; in fact, it turns out that all others are expressible in terms of these:

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

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

3.3 Operations 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 v_i in $(-)$ by the corresponding u_i, first alpha-converting $(-)$ to change any bound uses of $\vec{u}$. We shall use $\frown$ 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:

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

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 of the bodies, since the convention ensures that $S \frown T = T \frown S$ 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 $S \frown \sigma T \neq \sigma T \frown S$.

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. We shall also use the following non-functorial form of abstraction:

$$(x)a \stackrel{\text{def}}{=} (x\vec{u})S\langle\vec{v}\rangle \qquad (\text{where } a = (\vec{u})S\langle\vec{v}\rangle,\ x \notin \vec{u})\ .$$

3.4 Dynamics The atomic reaction relation $\searrow^1$ of PIC is defined as follows. Whenever a contains a subsequence like $\overline{u}\langle\vec{v}\rangle\, u(\vec{w})$, i.e. a takes the form

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

(after suitable commutations), then

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

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

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

■

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

One can check that $\searrow^1$ is indeed preserved by the operations; for example $a \searrow^1 a'$ implies $b \cdot a \searrow^1 b \cdot a'$.

The reaction relation $\searrow$ is defined to be $(\searrow^1)^*$, the transitive reflexive closure of the atomic relation.

3.5 Discussion PIC has considerable expressive power; for example it encodes the linear λ-calculus very naturally, mimicking β-reduction by reaction. The encoding is along the lines of [6]. But PIC needs extension to become a practically useful calculus of processes. This extension is done in [9], along uniform lines set out in [7]. But in that approach the resulting process calculus is not itself an action structure. By contrast, the extensions to PIC in Section 5 below given essentially the same expressive power, while remaining entirely within the action structure framework.

4 Controls and action calculi

In this section we see how to generalise PIC to a class of action structures which we shall call *action calculi*. The key to this generalisation is to observe that each particle of PIC is a special case of a richer construction which we shall call a *molecule*. This notion of molecule is not far from that of Berry and Boudol [1]. The molecules of any action calculus are formed from generators which we shall call *controls*.

4.1 Control constants Each particle π of PIC has some free and some binding occurrences of names. In 3.2 we used the symbol $\boldsymbol{\nu}$ and the indexed symbols $\mathbf{in}_x$, $\mathbf{out}_x$ for actions containing a single particle (of each of the three kinds). But this notation is somewhat ad hoc; it conceals a regularity which will lead us to a generalization of PIC, and indeed to a generalization of the π-calculus.

Consider the action $\langle\vec{y}\rangle \cdot \mathbf{out}_x$, with arity $0 \to 0$. In our partial sequence notation, it is the singleton $[\, x\langle\vec{y}\rangle \,]$. But there is no formal reason to treat the index x differently from the name-vector $\vec{y}$. We reveal the generality of particle formation more clearly if we simply declare that there are (for PIC) three *control constants*

$$\begin{array}{rl} \mathbf{in} & : 1 \to m \\ \mathbf{out} & : 1+m \to 0 \\ \boldsymbol{\nu} & : 0 \to 1 \ , \end{array}$$

and that for any control constant $C : m \to n$ there are PIC particles of the form

$$\langle\vec{y}\rangle C(\vec{x}) \qquad (|\vec{y}| = m,\ |\vec{x}| = n)$$

where the names $\vec{x}$ are distinct and binding. Thus our three forms of PIC particle $x(\vec{y})$, $\overline{x}\langle\vec{y}\rangle$ and $\boldsymbol{\nu} x$ are more truly written

$$\langle x\rangle \mathbf{in}(\vec{y}) \ ,\ \langle x\vec{y}\rangle \mathbf{out}(\,) \ ,\ \langle\,\rangle \boldsymbol{\nu}(x) \ .$$

Moreover, let us define

$$\begin{array}{rcl} \mathbf{out}_x : m \to 0 & \stackrel{\text{def}}{=} & (\langle x\rangle \otimes \mathbf{id}_m) \cdot \mathbf{out} \\ \mathbf{in}_x : 0 \to m & \stackrel{\text{def}}{=} & \langle x\rangle \cdot \mathbf{in} \ ; \end{array}$$

then the atomic reaction relation for PIC given in 3.4 can equally be defined by the rule

$$\mathbf{out}_x \otimes \mathbf{in}_x \searrow^{1} \mathbf{id}_m \ .$$

In a similar way we may define a wide variety of action structures, each one determined by a given set $\mathcal{K}$ of control constants and a set of reaction rules which determine their meaning. We shall call each action structure built in this way an *action calculus* and denote it by AC($\mathcal{K}$), understanding that with $\mathcal{K}$ the reaction rules are also given. Thus PIC=AC$\{\boldsymbol{\nu}, \mathbf{in}, \mathbf{out}\}$ is just a special case of this uniform construction.

This class of action calculi is still too restrictive; they all lack an important control feature. In these calculi, if $a \searrow a'$ is a possible reaction, then it may occur in any context; there is no way to delay it until some other activity is complete, or to make it conditional on the outcome of that activity. We now seek to remove this weakness.

4.2 Controls We wish to generalise the notion of control constant to allow activity of subactions to be constrained, modified or delayed. An example of delay is the simple guarding construction $a.P$ of CCS; P cannot act until a has happened. Sequential composition $P;Q$ in CSP is another example; Q cannot act until P has finished acting. So it is a natural step to allow action constructors C which take actions as arguments; we shall refer to these as *control constructors*, or more briefly as *controls*.

For each control C and suitable actions $a_1, \ldots, a_r$ there is a *control construction*

$$C(a_1, \ldots, a_r)$$

which is also an action. Each C is equipped with an arity rule of the form

$$\frac{a_1 : m_1 \to n_1 \quad \cdots \quad a_r : m_r \to n_r}{C(a_1, \ldots, a_r) : m \to n} \;(\chi)$$

where the side-condition χ may constrain the values of r, m_i, n_i, m, n. For example χ may determine that r is fixed, and then we say that C has *rank* r; a control of rank 0 is just a control constant, and we then omit the brackets in the construction $C(\,)$.

We shall look at one or two examples of controls in Section 5.

4.3 Action calculi: statics We are now ready to define the statics of the *action calculus* $\mathsf{AC}(\mathcal{K})$, where $\mathcal{K}$ is a given collection of controls with associated arity rules.

First, if $C\vec{a}$ is a control construction of arity $m \to n$, then

$$\langle\vec{x}\rangle\, C\vec{a}\,(\vec{y}) \qquad (|\vec{x}| = m, |\vec{y}| = n)$$

is a *molecule*. The names $\vec{y}$ must be distinct, and are *binding* occurrences; they provide the means by which the action $C\vec{a}$ can export data, just as the names $\vec{x}$ are data imported to the action.

The rest is just as in PIC, but with molecules μ instead of particles π. An action $a : m \to n$ now takes the form

$$(\vec{x})\,[\,\mu_1 \ \ldots \ \mu_k\,]\,\langle\vec{y}\rangle \qquad (|\vec{x}| = m, |\vec{y}| = n)\ ;$$

the body is again a partial sequence, i.e. we allow the commutation $\mu\mu' = \mu'\mu$ of a neighbouring pair of molecules provided that neither of them binds a name occurring in the other. The imported names $\vec{x}$ are bound. The scope of each bound name extends to the right of its binding occurrence, to the end of the smallest action containing it. Thus if a is an action argument within a molecule, the scope of any name bound within a is confined to a; however, any name free in a is also free in the molecule. As in PIC, alpha conversion is allowed.

4.4 Action calculi: dynamics As we have already suggested, the atomic reaction relation in an action calculus may be given by rules such as $\mathbf{out}_x \otimes \mathbf{in}_x \overset{1}{\searrow} \mathbf{id}_m$, which we shall call *control rules*. We understand that $\overset{1}{\searrow}$ is the smallest relation which satisfies the control rules and is closed under product, composition and abstraction.

Arbitrary complex control mechanisms may be introduced into action calculi; by merely defining the notion of action calculus we unfortunately do not succeed in isolating a distinguished family of controls! What we can claim is that the action calculus framework saves us much repetitious work in setting up a calculus for interaction; we can also hope that it allows us more readily to compare and classify such calculi, on the basis of the complexity of their controls and associated rules.

5 Examples of action calculi

In this section we first summarise the constituents of PIC, which is just $\mathsf{AC}(\nu, \mathbf{in}, \mathbf{out})$. This is the *essential* π-calculus, as it contains no more than the basic controls for restriction and name-passing. Then we enumerate a sequence of π-calculi, $\mathsf{PIC}(\mathcal{K})$, gained by adjoining extra controls $\mathcal{K}$ to PIC. We consider three controls: *boxing* (which may also be called *guarding*), *choice* and *replication*. These all come into π-calculus as defined in [10, 8], but we do not try to recover exactly their "standard" forms; our new framework presents control in a new light, and slight variations of meaning arise if we seek the simplest reaction rules.

We conclude the section by outlining an action calculus of Petri nets. Unless otherwise stated, we assume that the arities are natural numbers.

5.1 Basic π-calculus: $\mathsf{PIC} = \mathsf{AC}(\nu, \mathbf{in}, \mathbf{out})$

Controls :	$\nu,\ \mathbf{in},\ \mathbf{out}$ (rank 0)
Arity rules :	$\nu\ : 0 \rightarrow 1$ $\mathbf{in}\ : 1 \rightarrow m \qquad \mathbf{out}\ : 1+m \rightarrow 0$
Derived controls :	$\mathbf{out}_x : m \rightarrow 0 \stackrel{\text{def}}{=} (\langle x \rangle \otimes \mathbf{id}_m) \cdot \mathbf{out}$ $\mathbf{in}_x : 0 \rightarrow m \stackrel{\text{def}}{=} \langle x \rangle \cdot \mathbf{in}$
Control rule :	$\mathbf{out}_x \otimes \mathbf{in}_x \searrow^{1} \mathbf{id}_m$

One might have expected the control rule to be stated in the form

$$\langle \vec{y} \rangle \cdot \mathbf{out}_x \otimes \mathbf{in}_x \searrow^{1} \langle \vec{y} \rangle \ ;$$

but in fact these two rules generate the same atomic reaction relation, because of the closure conditions.

5.2 π-calculus with boxing: $\mathsf{PIC}(\mathbf{box})$

Control :	$\mathbf{box}$ (rank 1)
Arity rule :	$\dfrac{a\ : m \rightarrow n}{\mathbf{box}\, a\ : 1 \rightarrow n}$
Derived control :	$\mathbf{box}_x a : 0 \rightarrow n \stackrel{\text{def}}{=} \langle x \rangle \cdot \mathbf{box}\, a$
Control rule :	$\mathbf{out}_x \otimes \mathbf{box}_x a \searrow^{1} a$

It is clear that **in** $: 0 \to m$ is redundant in the presence of **box**, as it is essentially **box id**$_m$.

This calculus correspond to allowing *input* guards, but not *output* guards, in the π-calculus. From the work of Honda and Tokoro [4] we know that output guarding can in fact be defined in terms of input guarding. Written in the style of the reduction rules for π-calculus in [7], Section 2.4, the rule would appear as follows:

$$\overline{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \longrightarrow \{\vec{z}/\vec{y}\}P \ .$$

5.3 π-calculus with boxing and choice: PIC(**box**, **choose**)

Control :	**choose**
Arity rule :	$\dfrac{a_1 \ : 0 \to n \qquad \cdots \qquad a_r \ : 0 \to n}{\mathbf{choose}(a_1, \ldots, a_r) \ : 0 \to n}$
Control rule :	$\mathbf{out}_x \otimes \mathbf{choose}(\ldots, \mathbf{box}_x a, \ldots) \searrow^1 a$

This rule, written in the style of the π-calculus COMM rule ([8], 2.4) would appear as

$$\overline{x}\langle\vec{z}\rangle \mid (\cdots + x(\vec{y}).P + \cdots) \longrightarrow \{\vec{z}/\vec{y}\}P \ ;$$

thus it corresponds to a special case of the COMM rule, allowing choice and guarding for inputs but not for outputs. The reader should have no difficulty in defining controls to match the general case of COMM.

5.4 π-calculus with replication: PIC(**rep**)

Control :	**rep**	(rank 1)
Arity rule :	$\dfrac{a \ : m \to 0}{\mathbf{rep}\, a \ : 1 \to 0}$	
Derived control :	$\mathbf{rep}_x a \ : 0 \to 0 \ \stackrel{\text{def}}{=} \ \langle x\rangle \cdot \mathbf{rep}\, a$	
Control rule :	$\mathbf{out}_x \otimes \mathbf{rep}_x a \searrow^1 a \otimes \mathbf{rep}_x a$	

This form of replication differs from that described in [8], which was written $!P$. The replication of $\mathbf{rep}_x a$ is controlled by an interaction at x, as is clear from its control rule. An even clearer (equivalent) form of the rule is

$$\langle\vec{z}\rangle \cdot \mathbf{out}_x \otimes \mathbf{rep}_x a \searrow^1 \langle\vec{z}\rangle \cdot a \otimes \mathbf{rep}_x a \ ,$$

which shows clearly that a copy of the replicated action is "spun off" with parameters $\vec{z}$ received via x. The rule corresponds to a special case of reduction as defined in [8], namely

$$\overline{x}\langle\vec{z}\rangle \mid \ !x(\vec{y}).P \longrightarrow \{\vec{z}/\vec{y}\}P \mid \ !x(\vec{y}).P \ .$$

Our use of a control rule contrasts with the way replication is treated in [8]; there, a structural congruence law $!P \equiv P \mid !P$ is imposed, so that (without any special reduction rule) $!P$ is treated as arbitrarily many copies of P already coexisting. Such a law would break our present convention that controls are given meaning only by their reactions, not by axioms. There is also a related reason to prefer the present treatment – namely that it corresponds more closely to what would happen in a machine. Since an action calculus aims to be a rather concrete model of reaction, this correspondence is appropriate.

These extensions of PIC can of course be combined into PIC(**box**, **choose**, **rep**); for practical purposes this appears to be as expressive as any existing version of the calculus.

5.5 An action calculus for Petri nets: NETC = AC(ν, mark, pre, post)

We shall present, as an action calculus NETC, one of the action structures for Petri nets which were outlined in [7]. It is essentially an algebra – with dynamics – of place-transition nets (PT nets) which are parametric in both their places and their transitions. This parametrization allows two places or transitions in different nets, or even in the same net, to be coalesced with one another.

The monoidal structure of Petri nets was first exposed by Meseguer and Montanari [5]; that idea is essential to what follows. What is extra here is the power brought to the treatment by the composition and parametrization of action structures.

To begin with, we declare that the name-set X is partitioned into *place-names* $p, q, \ldots$ and *transition-names* $t, u, \ldots$. To reflect this duality properly we require the monoid of arities to consist of *pairs* of natural numbers, so that the unit is 0,0; a place-name p has arity $1, 0$ while a transition-name t has arity $0, 1$. The corresponding data actions have arities as follows:

$$\langle p \rangle : 0,0 \to 1,0 \qquad \langle t \rangle : 0,0 \to 0,1 \ .$$

The controls of NETC are very simple; apart from restriction they correspond to marking a place, and declaring a place to be either a pre-condition or a post-condition of a transition. We now give the constituents of NETC, and then discuss them.

Controls :	$\boldsymbol{\nu}$, **mark**, **pre**, **post** (rank 0)
Arity rules :	$\boldsymbol{\nu} : 0,0 \to 1,0 \quad \boldsymbol{\nu} : 0,0 \to 0,1 \quad \mathbf{mark} : 1,0 \to 0,0$ $\mathbf{pre} : 1,1 \to 0,0 \quad \mathbf{post} : 1,1 \to 0,0$
Derived controls :	$\mathbf{mark}_{\vec{p}} \stackrel{\text{def}}{=} \langle p_1 \rangle \cdot \mathbf{mark} \otimes \cdots \otimes \langle p_k \rangle \cdot \mathbf{mark} \quad (\lvert\vec{p}\rvert = k)$ $\mathbf{pre}^t_{\vec{p}} \stackrel{\text{def}}{=} \langle p_1 t \rangle \cdot \mathbf{pre} \otimes \cdots \otimes \langle p_k t \rangle \cdot \mathbf{pre} \quad (\lvert\vec{p}\rvert = k)$ $\mathbf{post}^t_{\vec{q}} \stackrel{\text{def}}{=} \langle q_1 t \rangle \cdot \mathbf{post} \otimes \cdots \otimes \langle q_k t \rangle \cdot \mathbf{post} \quad (\lvert\vec{q}\rvert = k)$ $\mathbf{trans}_{\vec{p};\vec{q}} \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)(\mathbf{pre}^t_{\vec{p}} \otimes \mathbf{post}^t_{\vec{q}})$
Control rule :	$\mathbf{mark}_{\vec{p}} \otimes \mathbf{trans}_{\vec{p};\vec{q}} \searrow^1 \mathbf{trans}_{\vec{p};\vec{q}} \otimes \mathbf{mark}_{\vec{q}}$

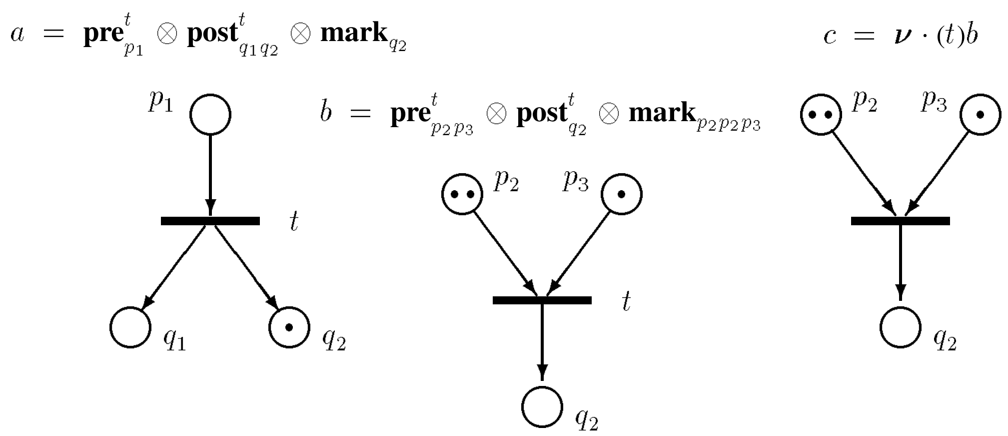

Figure 1: Three PT nets and their expression as actions

We shall first consider actions of arity $0, 0 \to 0, 0$; they are just PT nets in which some or all of the places and transitions bear names, each name occurring at most once in a net. An action has one particle for each arc of the net which it represents, and one for each token. Figure 1 shows three examples. The expressions for the first two nets, a and b, should be clear on inspection of the definition of derived controls. Note particularly that *every* place and transition is named in these two nets.

Now compare b with the third net c, whose transition is not named. The name has been hidden by composing restriction, $\boldsymbol{\nu}$, with an abstraction.

In passing, we note that abstraction may also be used for renaming. Thus the net $b' = \langle p_2 \rangle \cdot (p_3)b$ (not shown) is like b except that place p_3 is coalesced with place p_2; thus in b' the place p_2 is triply marked, and is a precondition of weight 2 to transition t.

Returning to the distinction between b and c, we observe that

$$c = \mathbf{mark}_{p_2} \otimes \mathbf{mark}_{p_2 p_3} \otimes \mathbf{trans}_{p_2 p_3 ; q_2} .$$

Thus, according to the control rule, c has a reaction as follows:

$$c \searrow^{1} \mathbf{mark}_{p_2 q_2} \otimes \mathbf{trans}_{p_2 p_3 ; q_2} ;$$

a token has been removed from each of p_2, p_3 and one placed on q_2, exactly as is dictated by the conventional net firing rule. On the other hand no reaction is possible for b. The reason is to do with the algebraic net constructions. In particular, the rule for forming a product requires like-named places and transitions to be coalesced. Thus a net must be seen only as a *partial* description of the pre- and post-conditions on its named transitions, since a coalescence can add further conditions. It is only when the name of a transition is restricted (i.e. removed from the net-diagram) that we can take the description to be *complete*, and thus know when it may fire. This is emphasized by the two products $a \otimes b$ and $a \otimes c$, which are illustrated in Figure 2. In the first case two transitions have been coalesced – and the resulting single transition cannot fire; but in the second case the firing condition on c's unnamed transition remains satisfied.

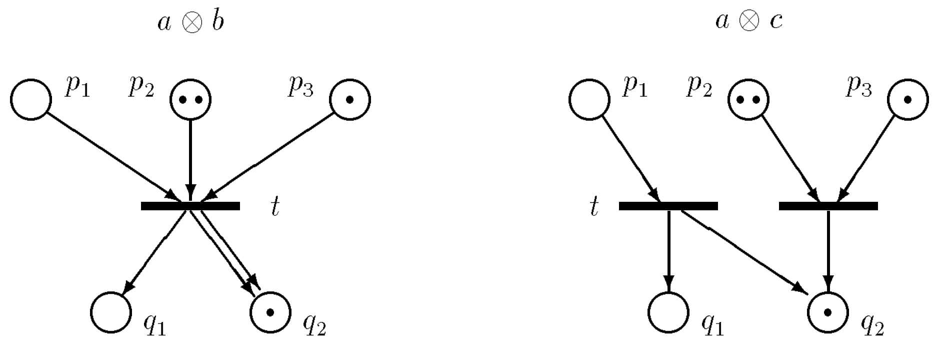

Figure 2: The products $a \otimes b$ and $a \otimes c$ of PT nets

Space prevents us from discussing the constructions of this action structure further, and we refer to [7] for more details. But several points arise even from what has been said. First, it seems clear that the notion of restriction $\boldsymbol{\nu}$ is of general significance, not merely confined to the π-calculus. Second, it seems worth pursuing the theory of Petri nets enriched by parametrization; the algebraic treatment which this allows appears remarkably natural, and can be based on just four very basic control constructions. Third, it is worth considering how to combine the basic ideas of net theory with the dynamic reconfiguration suggested by the π-calculus, within the single framework of action calculi. The form of the control rule in NETC ensures that a net remains unchanged after the firing of a transition, as is standard in net theory; but in the context of action calculi it is easy to propose variants of the rule which allow some reconfiguration to occur.

Finally, it is striking that only control *constants* (controls with no action parameters) are needed to define nets. Yet, informally speaking, nets appear to lack no power in representing computational phenomena. This suggests that careful study is needed to determine in what sense parametric controls add expressive power to action calculi.

6 Action calculi as free constructions

We shall now give an alternative characterisation for the statics of action calculi, which reveals clearly their algebraic status.

The characterisation is this: For each set $\mathcal{K}$ of controls, the action calculus $\mathsf{AC}(K)$ is isomorphic to the quotient of a term algebra by a certain congruence. The terms are generated from certain constants, by the basic action structure constructions and the control constructions; the congruence is induced by the basic action structure axioms together with further axioms. The latter, which we shall call the *concrete* axioms, do not vary from one action calculus to another since they do not mention the controls $\mathcal{K}$; the latter are truly *free* constructors, and their meaning lies entirely in their dynamics.

Throughout this section we continue to assume a fixed name-set X, and to assume that the arities are the natural numbers, with $x : 1$ for each $x \in X$. We also assume a fixed but arbitrary control set $\mathcal{K}$.

6.1 Action terms We shall use $t, u, \ldots$ to stand for *action terms*; the set $\mathcal{T}(\mathcal{K})$ of action terms is the smallest closed under the following term constructions:

$\boldsymbol{\omega} : 1 \to 0$	*discard*
$\langle x \rangle : 0 \to 1$	*datum* $(x \in X)$
$C\vec{t}$	control constructions $(C \in \mathcal{K})$
$\mathbf{id}_k,\ t \cdot u,\ t \otimes u,\ \mathbf{ab}_x t$	basic constructions.

The constructions are subject to rules of arity.

We find it convenient to define the following form of abstraction, which is closer to λ-abstraction:

$$(x)t \stackrel{\text{def}}{=} \mathbf{ab}_x t \cdot (\boldsymbol{\omega} \otimes \mathbf{id}_n) \qquad (t : m \to n).$$

Note that the postcomponent $\boldsymbol{\omega} \otimes \mathbf{id}_n$ has arity $1 + n \to n$; intuitively, it just discards its first datum. It will turn out that, in the presence of our axioms, $\mathbf{ab}_x$ is expressible in terms of (x); the axiomatization is a little smoother in terms of the latter.

6.2 Concrete axioms The concrete axioms are six in number. The first four characterize abstraction as a name-binding operation. Two of these correspond closely to the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ rules of λ-calculus; the other two, $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$, give something like the $\boldsymbol{\eta}$ rule of λ-calculus. To state these axioms we must first define when a name x occurs *free* in a term. This is standard, once we declare for the basic constants that x is free in $\langle y \rangle$ iff $x = y$, and no name is free in $\boldsymbol{\omega}$ or in $\mathbf{id}_k$. Then the substitution operation $\{y/x\}$, which changes bound names to avoid clashes, is also standard.

$$\begin{array}{lll}
\boldsymbol{\alpha} : & (x)t = (y)\{y/x\}t & (y \text{ not free in } t) \\
\boldsymbol{\beta} : & (\langle y \rangle \otimes \mathbf{id}_m) \cdot (x)t = \{y/x\}t & (t : m \to n) \\
\boldsymbol{\gamma} : & (x)t = \boldsymbol{\omega} \otimes t & (x \text{ not free in } t) \\
\boldsymbol{\delta} : & (x)(t \otimes \mathbf{id}_m) = (x)t \otimes \mathbf{id}_m \; . &
\end{array}$$

We now need two further abbreviations:

$$\begin{array}{rcl}
\langle x_1 \cdots x_k \rangle & \stackrel{\text{def}}{=} & \langle x_1 \rangle \otimes \cdots \otimes \langle x_k \rangle \\
(x_1 \cdots x_k)t & \stackrel{\text{def}}{=} & (x_1) \cdots (x_k)t \; .
\end{array}$$

Using these, we define a family of permutation actions:

$$\mathbf{i}_{km} \stackrel{\text{def}}{=} (\vec{x}\vec{y})\langle \vec{y}\vec{x} \rangle \qquad (|\vec{x}| = k, |\vec{y}| = m) \; .$$

Note that $\mathbf{i}_{km} : (k \otimes m) \to (m \otimes k)$ is an isomorphism of arities; from the axioms one can show that $\mathbf{i}_{mk}$ is indeed its inverse. In fact, under composition the $\mathbf{i}_{km}$ generate all and only the action structure isomorphisms.

Of the two remaining axioms, $\boldsymbol{\epsilon}$ says that the trivial permutation $\mathbf{i}_{10}$, which is just $(x)\langle x \rangle$, is the unit of arity 1; $\boldsymbol{\zeta}$ says that permutation of arity agrees with permutation of the product of actions:

$$\begin{array}{ll}
\boldsymbol{\epsilon} : & \mathbf{i}_{10} = \mathbf{id}_1 \\
\boldsymbol{\zeta} : & \mathbf{i}_{km} \cdot (u \otimes t) = (t \otimes u) \cdot \mathbf{i}_{ln} \quad (t : k \to l,\ u : m \to n) \; .
\end{array}$$

To give a feeling for the power of these axioms, together with the basic action structure axioms, we list a few of their consequences; the proofs are routine.

6.3 Lemma From the action structure axioms on terms, together with the concrete axioms, it follows that

$$\begin{array}{ll}
(x)(t\cdot u) = (x)t\cdot u & (x \text{ not free in } u)\\
(x)(t\cdot u) = (\mathbf{id}_1\otimes t)\cdot(x)u & (x \text{ not free in } t)\\
(x)(t\otimes u) = (x)t\otimes u & (x \text{ not free in } u)\\
(x)(t\otimes u) = t\otimes(x)u & (t : 0\to n,\ x \text{ not free in } t)\\
\mathbf{i}_{k0} = \mathbf{i}_{0k} = \mathbf{id}_k & \\
\mathbf{ab}_x t = (x)(\langle x\rangle\otimes t)\ . &
\end{array}$$

■

The last equation justifies our claim that $\mathbf{ab}_x$ and (x) are interdefinable.

Clearly $\mathcal{T}(\mathcal{K})/\equiv$ is an action structure, where $\equiv$ is the congruence induced by the basic and concrete axioms. Now recall that in [7] a (standard) notion of homomorphism and isomorphism was defined for action structures. With the help of the lemma we get the following significant result:

6.4 Theorem $\mathcal{T}(\mathcal{K})/\equiv$ is isomorphic to $\mathsf{AC}(\mathcal{K})$. ■

Such a result is not surprising; we would expect $\mathsf{AC}(\mathcal{K})$ to be characterized by *some* set of equational axioms. Nonetheless the axioms $\boldsymbol{\alpha}$–$\boldsymbol{\zeta}$ have some intuitive appeal.

We have been somewhat informal in our notation. Strictly, we should have adopted different notations for the constructions of $\mathcal{T}(\mathcal{K})$, such as $\boldsymbol{\omega}$, $\langle x\rangle$ and $(x)(-)$, from those already used in $\mathsf{AC}(\mathcal{K})$. But in using notation ambiguously we have merely anticipated the isomorphism which is assured by the theorem. The isomorphism also assures us that the partial sequence notation of Section 4 can be mixed with the term language with impunity; for example if the names $\vec{x}, \vec{y}$ do not occur free in $\vec{a}$ then

$$(\vec{x})\,[\,\langle\vec{x}\rangle\, C\vec{a}\,(\vec{y})\,]\,\langle\vec{y}\rangle \;=\; C\vec{a}\ ,$$

where the left-hand form is in partial sequence notation, and the right-hand term is in $\mathcal{T}(\mathcal{K})$.

6.5 Concrete action structures Let us look more closely at the concrete axioms. We have used them just to impose a congruence upon the terms $t, u, \ldots$ of $\mathcal{T}(\mathcal{K})$. But let us now regard them as axioms which may or may not be satisfied by the actions $a, b, \ldots$ of an arbitrary action structure:

	CONCRETE AXIOMS	
$\boldsymbol{\alpha}$:	$(x)a = (y)\{y/x\}a$	(y not free in a)
$\boldsymbol{\beta}$:	$(\langle y\rangle \otimes \mathbf{id}_m) \cdot (x)a = \{y/x\}a$	$(a : m \to n)$
$\boldsymbol{\gamma}$:	$(x)a = \boldsymbol{\omega} \otimes a$	(x not free in a)
$\boldsymbol{\delta}$:	$(x)(a \otimes \mathbf{id}_m) = (x)a \otimes \mathbf{id}_m$	
$\boldsymbol{\epsilon}$:	$\mathbf{i}_{10} = \mathbf{id}_1$	
$\boldsymbol{\zeta}$:	$\mathbf{i}_{km} \cdot (b \otimes a) = (a \otimes b) \cdot \mathbf{i}_{ln}$	$(a : k \to l,\ b : m \to n)$.

But we are now dealing with a richer algebraic variety than action structures, since the axioms are phrased in terms of the predicate “free in” $\subseteq X \times A$ and the renaming operation $\{y/x\} : A \to A$.

A variety can indeed be defined with such additional operations, which are subject to some natural laws. Let us call this variety the *concrete action structures*. Then it turns out that our action calculus $\mathsf{AC}(\mathcal{K})$ is indeed initial in the appropriate category of concrete action structures over $\mathcal{K}$; that is, action calculi are the *free* concrete action structures.

7 Future work

A first line for further investigation is to compare the way an action calculus is extended to a larger one, by adding controls, with the approach taken in [7] where it is shown how to define a process calculus (which is *not* an action calculus) on top of an arbitrary action structure. The latter approach aimed specifically at recovering the semantic congruence of bisimilarity between processes in a uniform manner. To do so, it exploited two notions which appear to be natural refinements of an action structure – the notions of *effect structure* and *incident set*. It is not yet clear how to treat bisimilarity, with or without these notions, in an action calculus; this is a vitally important task, since it is essential to divide out by a larger congruence in order to treat action calculi semantically.

More generally, the way is now open to attempt a classification of concrete models of concurrency, in terms of the controls which characterize them as action calculi. An understanding of the relative power of various control mechanisms is notably absent.

Finally, the class of concrete action structures mentioned at the end of Section 6 deserves further study.

References

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

[2] Banâtre, J.P. and Métayer, D., *The GAMMA model and its discipline of programming*, Science of Computer Programming, Vol 15, pp55–77, 1990.

[3] Hoare, C.A.R., *Communicating Sequential Processes*, Communications of ACM, Vol 21, pp666–677, 1978.

[4] Honda, K. and Tokoro, M., *An object calculus for asynchronous communication*, Proc. European Conference on object-oriented programming, Lecture Notes in Computer Science, Vol 512, Springer, pp133–147, 1991.

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

[6] Milner, R., *Functions as processes*, Math. Struct. in Comp. Science, Vol 2, pp119–141, 1992.

[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 ECS–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., *Action structures for the π-calculus*, Research Report ECS–LFCS–93–264, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992.

[10] 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.

[11] Petri, C.A., Fundamentals of a theory of asynchronous information flow, Proc. IFIP Congress '62, North Holland, pp386–390, 1962.