Calculi for Interaction

Robin Milner
Cambridge, April 1995

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. Each action in an action calculus is represented as an assembly of *molecules*; the syntactic binding of *names* is the means by which molecules are bound together. A graphical form, *action graphs*, is used to aid presentation. One action calculus differs from another only in its generators, called *controls*.

Action calculi generalise a previously defined action structure PIC for the π-calculus. Several extensions to PIC are given as action calculi, giving essentially the same power as the π-calculus. An action calculus is also given for the typed λ-calculus, and for Petri nets parametrized on their places and transitions.

An equational characterization of action calculi is given: each action calculus A is the quotient of a term algebra by certain equations. The terms are generated by a set of operators, including those basic to all action structures as well as the controls specific to A; the equations are the basic axioms of action structures together with four additional axiom schemata.

1 Introduction

Background There is a remarkable variety among basic calculi for computation. Perhaps the most familiar, at least the most commonly known as a "calculus", is the λ-calculus. But other operational models – Turing machines, register machines, recursion equations, … – also deserve to be called calculi; in each case there is a formalism, and some rules and metatheory about the construction and combination of terms in the formalism. The variety is such that we cannot claim a clear understanding of the *family* of all such calculi, even though an underlying theory of functions has been developed which gives them semantic unity.

When "computation" is expanded to include concurrent or interactive behaviour, we cannot rest only upon the intuitions of such a theory in formulating a calculus. As a consequence, there is even greater variety among calculi which describe interactive systems. (Examples are: Petri nets, many process algebras, communicating automata, statecharts, ….) So perhaps we have even less hope of classifying them. Yet it may not be so; since the population of calculi is larger, repeated features among them may be more apparent. By giving attention to *interaction* as a fundamental notion, we may reveal regularities not hitherto detected.

The need for some classifying discipline is all the greater, since the generalisation from computation to interaction is also a move from prescriptive to descriptive. In the world of networks and distributed computing we cannot claim that all the systems we study are built to a prescription; instead, we seek a descriptive theory to analyse the phenomena of an ever more complex informatic world, which is partly "natural" and partly man-made, and in which computation is a special case. A taxonomy of models of interaction will be a step towards such a theory.

Motivation and character In this paper a mathematical framework is proposed for studying, comparing and combining operational models of interaction. Each such model appears in the framework as an *action calculus*; action calculi are a special class of *action structures* [15], and each member of the class is determined by its generators, called *controls*. Two different characterizations of action calculi are given, and several examples of action calculi are presented.

The paper prepares the way for the common treatment of the semantic interpretations of all action calculi, via homomorphisms of action structures; this will be pursued in later papers. One particular aim is to find a general treatment of behavioural equivalences such as bisimilarity. Another aim is to classify action calculi according to their dynamic qualities; for example, the π-calculus has been called a calculus of *mobile* processes, and we would like to say precisely what is meant by mobility.

The next few paragraphs give some general motivation for action calculi, and roughly explain their character.

If we consider Petri nets [20], CSP [7] and the π-calculus [19] beside the λ-calculus [2], we find recurrent features. The parallel composition of CSP looks like the juxtaposition (with some transition-sharing) of Petri nets; exact translations between the two have been made on this basis. The π-calculus has binding of names, begging comparison with the richer notion of bound variable in the λ-calculus. Parallel reduction of a term of the λ-calculus, in the presence of a shared valuation of the free variables, is somewhat like the interaction between several independent agents and a shared resource; such interactions are well-represented in Petri nets. Finally, the dynamic rules of such models are often expressed in a similar way: the reduction of, or reaction within, certain key control configurations (called *redexes* in λ-calculus, and *fireable transitions* in Petri nets).

Of these common features, the use of *names* is one of the most crucial, and presents a fundamental challenge. The problem is that names are used in different ways. In the λ-calculus they are variables which may be replaced by any value (or term denoting a value); in the π-calculus they are channels, and also variables – but only over channels; in Petri nets they are sometimes used to identify transitions (but not as variables over transitions). The solution adopted here is that names, even when variables, may only stand for names. Thus we have separated the two attributes of a name which are combined in λ-calculus (and often treated as inseparable!); a name may *vary*, and it may *denote a value*. in action structures we take *varying* as basic, and represent it by an abstraction operator; *denoting* can be treated as a special case of interaction, as will be seen in Section 5.5 where the λ-calculus is presented.

The basic ingredients of action structures – composition, tensor product, abstraction and a reaction relation – were chosen as a minimum to provide a uniform treatment of features common to many calculi. (The mathematical structure defined by these ingredients alone may be too weak to be of independent interest; action structures should be considered mainly as a basis for the enrichments introduced here.) In addition, action calculi provide a uniform way to introduce *specific* control mechanisms; behaviour is represented by interaction between the members of certain control configurations. The framework hardly limits the variety of such control disciplines; but it allows us to ask what happens if we combine them. For example, can we enrich Petri nets by allowing them to change configuration dynamically? We can formulate this enrichment precisely as the combination of the controls of two action calculi: Petri nets and the π-calculus.

The action structure framework was designed to accommodate not only the syntax (operational models) but also the semantics (abstract models) of interactive behaviour. In [15] was given an example expressing the meaning of a simple calculus as a homomorphism between two action structures, one expressing the evaluation of expressions and the other containing the denoted values. Since we wish to extend this treatment to calculi in general, we first have to determine the class of action structures which can be considered as calculi.

Technical overview A standard way to set up a calculus is to define a term algebra, and then provide it with reduction rules or transition rules. But for interactive systems it has been found helpful to use a formalism richer than a term algebra. Following ideas of Banâtre and Métayer [4], Berry and Boudol based their Chemical Abstract Machine (Cham) [3] on the notion of 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 [14]. Again, Meseguer and Montanari [10] have revealed the monoidal structure inherent in Petri nets. Indeed, part of the reason for the success of Petri nets is that its syntax, being graphical, reflects this structure.

Action structures impose monoidal structure. In fact they go further; they include the operation of abstraction (parametrization) over names, and also represent dynamics as a preorder. Both the monoidal and the additional structure are characterized by algebraic axioms.

Let us outline how an action calculus is defined as the quotient of a term algebra. We begin with the operators of action structures, and arrive at action calculi by three further steps. The first step is to introduce *naming constants*; these give power to an action structure to manipulate names, which act as the "wiring" or connective tissue for all action calculi. Four equational axioms called *naming axioms* are imposed upon the naming constants. This first step is common to all action calculi. The next step is specific to each action calculus: we supply a family of *control operators*, or *controls*, together with a set of *control rules* which define the reaction relation. The first two steps yield a term algebra defined by all these operators; the final step is to quotient this term algebra by the action structure and naming axioms. These quotient algebras also have an appealing concrete presentation; each action can be considered as a structure of molecules in the spirit of the Cham.

An action structure PIC, corresponding to a fragment of the π-calculus [16], was described in [15]; it now finds its place as an action calculus with just three generators; $\boldsymbol{\nu}$ (restriction), **in** (input) and **out** (output). An action calculus for Petri's place-transition nets is generated by $\boldsymbol{\nu}$, **pre** (pre-condition), **post** (post-condition) and **m** (marking). Thus action calculi begin to achieve one of the aims of action structures – to unite different models of concurrency in a common setting.

The action calculus concept goes beyond what was achieved in [15]. That paper failed to express a full-fledged process calculus as an action structure (PIC falls short of the full π-calculus, since it lacks the prefixing, summation and replication constructions); instead, it showed how to build a process calculus *on top of* an arbitrary action structure. We show here that this further structure is redundant; process calculi can be exhibited as action calculi.

Organisation of the text The main technical definitions and results[1] of the paper appear in Sections 4 and 6; the remaining sections supply background, motivation and examples. The paper is organised as follows. Section 2 reviews the basic definition of action structures. Section 3 reviews the action structure PIC, an exemplar for the general definition of action calculus. Section 4 defines action calculi formally in terms of their concrete presentation, molecular forms; a graphical

[1] The results in this paper were first announced, without proofs, in [17]. However, the six axioms of that paper have here been reduced to four.

presentation called *action graphs* is also defined. Section 5 gives a series of examples of action calculi; it first shows how PIC can be extended to richer action calculi by adding further generators, then presents both the typed λ-calculus and Petri nets as action calculi. Action graphs are freely used, especially to present control rules. Section 6 gives formally the algebraic characterization of action calculi; it demonstrates that each action calculus is isomorphic to the quotient of a term algebra by the action structure axioms together with four further axiom schemata. Section 7 identifies some further lines of investigation; in particular, it sketches the recently discovered *control structures* [12]. A control structure is an action structure with additional structure; each set of controls equipped with dynamic rules determines not only an action calculus but also a category of control structures in which the action calculus is initial; these control structures are therefore semantic interpretations of the action calculus.

Acknowledgements I owe much to intense discussions with Yoram Hirshfeld, around Christmas 1993. He helped me to explore many paths, attractive and thorny; many were cul-de-sacs, but the exploration contributed to the present ideas. I thank Adriana Compagnoni, Philippa Gardner, Ole Jensen, Benjamin Pierce, John Power and Peter Sewell, for many suggestions which have improved the paper.

2 Action structures reviewed

In this section we recall from [15] the basic notion of an action structure. We first give it in category-theoretic terms, then elaborate it algebraically. A little familiarity with the notion of monoidal category will help the reader, but no further knowledge of categories is needed.

2.1 Definition (Action structure) A (*dynamic*) *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.

If the preorder is the identity relation, or is not supplied, A is called a *static* action structure. ■

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.

An action structure A possesses, first, a monoid $(M_A, \otimes, \epsilon)$ of objects; these are the objects of the category and we shall call them *arities*. We shall use $k, \ell, m, n, \ldots$ 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. With each name x is associated an arity; if this is k, we write $x : k$. For a vector $\vec{x} = x_1 \cdots x_r$ of names with arities $k_1 \cdots k_r$, we write $\vec{x} : k$ where $k = k_1 \otimes \cdots \otimes k_r$.

For each pair m, n of arities A possesses, second, 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 $a, b, c, \ldots$ 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, the operations of composition $\cdot$, tensor or monoidal product $\otimes$, and an abstraction operator $\mathbf{ab}_x$ for each $x \in X_A$. They obey the following *arity rules*:

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

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

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

The status of A as static action structure 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:

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

Finally, in a dynamic action structure 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, i.e. $a \searrow a'$ implies $b \otimes a \searrow b \otimes a'$, $\mathbf{ab}_x a \searrow \mathbf{ab}_x a'$, etc. The identities are required to be minimal for the reaction relation, i.e. $\mathbf{id} \searrow a$ implies $a = \mathbf{id}$.

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 channel.

Abstraction $\mathbf{ab}_x$ allows parametrization upon the name x (not upon arbitrary values, as in λ-abstraction). When x is a channel 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. We may think of an action as an *activity*; then for $a : k \to \ell$ and $b : \ell \to m$ the arity ℓ describes an interface through which, in the composition $a \cdot b$, the activity of b may be influenced by that of a. We may think of this influence as *information* flowing across the interface, any time during the continuing activity of a and b. (Software terms for this are "dataflow" or "pipelining".)

We have emphasized how tensor product and composition differ in how they permit interaction, but they also have common features. Indeed this must be so, since any tensor product $a \otimes b$ is expressible as a composition $(a \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes b)$, and the identities $\mathbf{id}$ contribute nothing to interaction.

The examples of reaction in Section 3 will clarify these intuitions.

3 An action structure for the π-calculus

In this section we define an action structure PIC, corresponding to a fragment of the π-calculus. It is studied more fully in Part II of [15]. Here we shall use it as an exemplar and motivation for the definition of action calculi to be given in Section 4.

We take the arities of PIC to be $(\mathbf{N}, +, 0)$ – the natural numbers under addition.

3.1 Particles PIC formalises two basic features of the π-calculus; the passage of names through ports which are also names, and the localisation of names by restriction. The 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* $\vec{\pi}$ 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 adjacent pair, if neither binds a name occurring in the other. An action $a : m \to n$ is a *particle form*

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

where $\vec{x}$ is an m-vector of distinct names, the *imported* names, and $\vec{y}$ is an n-vector of names (not necessarily distinct), the *exported* names. The imported names in a particle form are bound. The scope of each bound name – either imported or bound by a particle – 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). For clarity, we may enclose a sequence $\vec{\pi}$ in square brackets.

As an example, let

$$a = (u)\,[\,\overline{u}\langle x\rangle\,]\,\langle u\rangle \ , \qquad b = (v)\,[\,v(w)\,\overline{w}\langle y\rangle\ x(z)\,]\,\langle z\rangle \ ;$$

then the composite $a \cdot b$ will impose the substitution $\{u/v\}$ upon the body of b, yielding

$$a \cdot b \ = \ (u)\,[\,\overline{u}\langle x\rangle\ u(w)\,\overline{w}\langle y\rangle\ x(z)\,]\,\langle z\rangle \ .$$

We now define all the action structure operations for PIC.

3.3 Operations First, the identities are given by

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

We shall apply substitutions like $\sigma = \{\vec{u}/\vec{v}\}$, where $\vec{v}$ is a vector of distinct names, to various syntactic forms F;; $_\sigma F$ is the result of simultaneously replacing each free occurrence of v_i in F by the corresponding u_i, first alpha-converting F to change any bound uses of $\vec{u}$. Note that only a name can replace a name; we do not use the general form of substitution in which arbitrary terms replace variables.

Composition, product and abstraction are given as follows, where we assume $a = (\vec{u})\,\vec{\pi}\,\langle\vec{v}\rangle$, $b = (\vec{x})\,\vec{\varrho}\,\langle\vec{y}\rangle$ and that neither binds a name occurring in the other:

$$\begin{array}{rcl} a \cdot b & \stackrel{\text{def}}{=} & (\vec{u})\,\vec{\pi}\,{}_\sigma\vec{\varrho}\,\langle{}_\sigma\vec{y}\rangle \quad (\sigma = \{\vec{v}/\vec{x}\}) \\ a \otimes b & \stackrel{\text{def}}{=} & (\vec{u}\vec{x})\,\vec{\pi}\,\vec{\varrho}\,\langle\vec{v}\vec{y}\rangle \\ \mathbf{ab}_x a & \stackrel{\text{def}}{=} & (x\vec{u})\,\vec{\pi}\,\langle x\vec{v}\rangle\ . \end{array}$$

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 $\vec{\pi}\vec{\varrho} = \vec{\varrho}\vec{\pi}$ in this case. But in $a \cdot b$ the substitution σ may replace some names free in $\vec{\varrho}$ by names bound in $\vec{\pi}$, and then $\vec{\pi}\,({}_\sigma\vec{\varrho}) \neq ({}_\sigma\vec{\varrho})\,\vec{\pi}$.

It is a routine matter to verify the action structure axioms.

3.4 Dynamics The reduction 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. whenever a takes the form

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

(after suitable commutations), we have

$$a \searrow^1 (\vec{x})\,\vec{\pi}\,{}_\sigma\vec{\varrho}\,\langle{}_\sigma\vec{y}\rangle,$$

where σ is the substitution $\{\vec{v}/\vec{w}\}$. We call $\overline{u}\langle\vec{v}\rangle\, u(\vec{w})$ a *redex* of a. As an illustration, continuing the example at the end of 3.2 we have

$$\begin{array}{rcl} a \cdot b & = & (u)[\,\overline{u}\langle x\rangle\, u(w)\,\overline{w}\langle y\rangle\, x(z)\,]\langle z\rangle \\ & \searrow^1 & (u)[\,\overline{x}\langle y\rangle\, x(z)\,]\langle z\rangle \\ & \searrow^1 & (u)\langle y\rangle\ . \end{array}$$

Note that $a \cdot b$ has two reductions, even though a and b alone have none. Note also that the port of a redex (in the example, first u then x) may be free or bound.

In contrast, consider:

$$a = (u)\,[\,\overline{u}\langle x\rangle\ u(v)\,]\,\langle v\rangle\ , \qquad b = (v)\,[\,\overline{w}\langle v\rangle\ w(z)\,]\,\langle z\rangle\ .$$

In this case, both a and b contain redexes, so we have $a \searrow^1 a' = (u)\,[\ \ ]\,\langle x\rangle$ and $b \searrow^1 b' = (v)\,[\ \ ]\,\langle v\rangle$. In $a \cdot b$ there is no requirement that a react before b, so we have two reaction sequences:

$$\begin{array}{ccccc} a \cdot b & \searrow^1 & a' \cdot b & \searrow^1 & (u)\,[\ \ ]\,\langle x\rangle \\ a \cdot b & \searrow^1 & a \cdot b' & \searrow^1 & (u)\,[\ \ ]\,\langle x\rangle\ . \end{array}$$

This illustrates the point made in 2.3 that composition represents dataflow, not sequential composition.

One can check that $\searrow^1$ is 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 reduction.

3.5 Discussion PIC has considerable expressive power; for example it encodes the linear λ-calculus naturally, mimicking β-reduction by reaction. The encoding is along the lines of [14]. But PIC needs to be extended if it is to be a practically useful calculus of processes. The extensions to PIC defined in Section 5 below provide at least the expressive power of the original π-calculus, while remaining entirely within the framework of action structures.

We shall now show how PIC, though apparently quite specific, is an instance of a more general construction. We begin by showing how all actions with empty bodies can be generated.

3.6 Naming actions We call the following actions *naming actions*:

$$\begin{aligned} \langle x\rangle : 0 \to 1 &\stackrel{\text{def}}{=} (\,)[\;\;]\langle x\rangle \\ \boldsymbol{\omega} : 1 \to 0 &\stackrel{\text{def}}{=} (x)[\;\;]\langle\,\rangle \;. \end{aligned}$$

It is easy to show that every action $a = (\vec{x})[\;\;]\langle\vec{y}\rangle$, with an empty body, can be expressed in terms of naming actions via the action structure operations as defined in 3.3. For example,

$$(xy)[\;\;]\langle yy\rangle = \boldsymbol{\omega} \otimes \mathbf{ab}_y\langle y\rangle \;.$$

These body-free actions are just concerned with wiring; they form the export vector from the import vector by copying, permuting and discarding.

3.7 Control constants Each particle π of PIC may have some free and some binding occurrences of names. Indeed, if we ignore dynamics, the only difference among the three kinds of particle is in the number of free and bound names they carry. We therefore reveal the generality of particle formation more clearly if we simply declare that there are (for PIC) three *control constants*

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

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

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

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

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

Note that the free names in each particle appear first, since the scope of the names *bound* by the particle must not include them.

In fact, we can associate with each control constant K an action $(\vec{x})[\,\langle\vec{x}\rangle K(\vec{y})\,]\langle\vec{y}\rangle$ containing just a single particle. Thus for PIC we define

$$\begin{aligned} \boldsymbol{\nu} &\stackrel{\text{def}}{=} (\,)[\,\langle\,\rangle\boldsymbol{\nu}(x)\,]\langle x\rangle \\ \mathbf{out} &\stackrel{\text{def}}{=} (x\vec{y})[\,\langle x\vec{y}\rangle\mathbf{out}(\,)\,]\langle\,\rangle \\ \mathbf{in} &\stackrel{\text{def}}{=} (x)[\,\langle x\rangle\mathbf{in}(\vec{y})\,]\langle\vec{y}\rangle \;. \end{aligned}$$

It is now easy to show that *every* action in PIC is expressible in terms of the naming actions and control constants, via the action structure operations.

Moreover, the dynamics of PIC is elegantly expressible in these terms. Let us define

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

then it can be verified that the reaction relation for PIC given in 3.4 is the smallest preorder $\searrow$ preserved by the action structure operations, and obeying the rule

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

Thus we see clearly what is specific to PIC: its set of control constants, their arities, and their dynamics as expressed by the above rule. 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 dynamic rules called *control rules* which determine their meaning. According to Definition 4.12 to follow, each action structure built in this way (with reaction rules also given for $\mathcal{K}$) is an *action calculus*, denoted by AC($\mathcal{K}$). PIC = AC($\boldsymbol{\nu}, \mathbf{out}, \mathbf{in}$) is just a special case of this uniform construction.

However, the action calculi generated in this way are not sufficiently general; they 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. Definitions 4.4 and 4.12 will remove this deficiency.

4 Controls and molecular forms

In this section we formally define a class of action structures which we shall call *action calculi*. Their actions are more general than the particle forms discussed in the preceding section; in fact a particle $\langle\vec{y}\rangle K(\vec{x})$ is a special case of a richer construction which we shall call a *molecule*. This notion of molecule is similar to that of Berry and Boudol [3]; one difference is that our molecules can bind one another, since a molecule is a name-binding operator. The molecules of any action calculus are formed from generators which we shall call *control operators*, or simply *controls*.

A control operator is a generalisation of the notion of control constant; its purpose is to control the activity of subactions. An example 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. Lambda abstraction in the lazy λ-calculus is a third example; the redex $(\lambda x M)N$ must be reduced before M is reducible.

An action calculus will be determined by a set $\mathcal{K}$ of controls, which we call a *signature*, together with a set $\mathcal{R}$ of control rules which we shall define later. We let K range over controls.

4.1 Definition (Control) A *control* K is an operator which allows the construction of an action $K(\vec{a})$ from a sequence $\vec{a}$ of actions, subject to a *rule of arity* having the following form:

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

where the side-condition χ may constrain the integer r and the arities m_i, n_i, m, n. If r is fixed, it is called the *rank* of K; otherwise we may say that K has *variable rank*. ■

An example of a signature is $\mathcal{K} = \{\boldsymbol{\nu}, \mathbf{out}, \mathbf{in}\}$; with appropriate arity rules and control rules it determines PIC, presented formally in Section 5.1. Its controls all have rank 0; that is, they are control constants. Another example is $\mathcal{K} = \{\boldsymbol{\lambda}, \mathbf{ap}\}$, where $\boldsymbol{\lambda}$ has rank 1; in Section 5.5 we see that it determines the λ-calculus (either simply typed or type-free, depending on the arity monoid).

From now on we assume a fixed denumerable name-set X. We also impose a constraint upon the monoid M of arities of an action calculus, bearing in mind its operational purpose. We require that M be freely generated by a set of *prime* arities $p, q, \ldots$, and that names be associated only with prime arities, infinitely many names with each prime.

We are now ready for our first presentation of action calculi, in terms of syntactic constructions known as *molecular forms*.

4.2 Definition (Molecules and molecular forms) Let $\mathcal{K}$ be a signature. The *molecular forms over* $\mathcal{K}$ are syntactic objects; they consist of the actions a defined as follows, in terms of *molecules* μ:

$$a \;::=\; (\vec{x})\,\vec{\mu}\,\langle\vec{u}\rangle \qquad (\vec{x} : m,\ \vec{u} : n,\ a : m \to n)$$

$$\mu \;::=\; \langle\vec{v}\rangle\, K\vec{b}\,(\vec{y}) \quad (\vec{v} : k,\ \vec{y} : \ell,\ K\vec{b} : k \to \ell)\ .$$

Molecules $\lambda, \mu, \ldots$ are binding operators. In the above molecule μ, the names $\langle\vec{v}\rangle$ occur free; they are the means by which it is bound into an action. In the above action a, any name-vector in round brackets – either at the head of a or at the right end of a molecule in $\vec{\mu}$ – is binding, and its scope extends rightwards to the end of a. Names which are not thus bound are free in a. We write $\mathbf{fn}\,a$ for the free names of a. Alpha-conversion of bound names is allowed. We assume that no name has more than one binding occurrence in any molecule or action.

In a above, $\vec{x}$ are called the *imported names* and $\vec{u}$ the *exported* ones. The *body* $\vec{\mu} = \mu_1 \cdots \mu_r$, sometimes written $[\,\mu_1, \ldots, \mu_r\,]$, is a possibly empty *partial* sequence of molecules, in which any two adjacent molecules may be commuted if neither binds a name occurring free in the other. ■

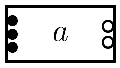
As an example, suppose $\mathcal{K}$ contains controls K_1, K_2, K_3 of rank $0, 1, 0$ respectively; then, under suitable rules of arity, a possible action is

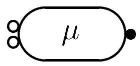
$$a \;=\; (x_1x_2x_3)\,[\ \langle x_1\rangle K_1\,(u_1u_2),\ \ \langle x_2x\rangle K_2 b\,(v),\ \ \langle u_2x_2v\rangle K_3\,(w)\ ]\,\langle u_1x\rangle$$
$$\text{where } b = (z)\langle u_2zy\rangle\ .$$

Note that in this case no adjacent pair of molecules can be commuted, since the first two bind names in their successors.

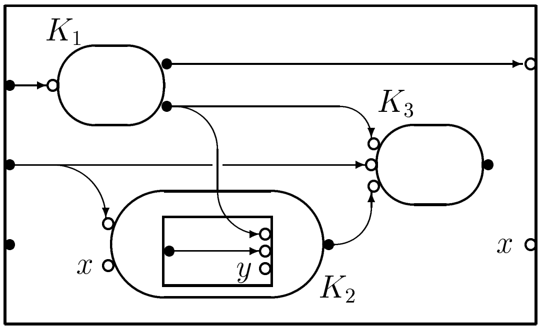
4.3 Action Graphs Molecular forms can be presented graphically by *action graphs*, which we now describe informally.

We shall use rectangles for actions and ovals for molecules. For an action a with arity $3 \to 2$ and a molecule μ whose control construction has arity $2 \to 1$ we draw respectively





Rectangles and ovals are nested alternately. A *source* (•) stands for a binding occurrence of a name; a *sink* (∘) stands for a bound or free occurrence. To each sink corresponding to a bound occurrence, an arc is drawn from the source which binds it; each free occurrence of a name is labelled by the name. Here is the graph which represents the example displayed above:



Such diagrams help in understanding action calculi; in particular, we shall use them to display control rules. The graphs can be treated with full rigour, and will be formally presented in Ole Jensen's forthcoming PhD thesis [9]. Indeed, they may be preferred to molecular forms for many purposes.

We now define the action structure operations over molecular forms.

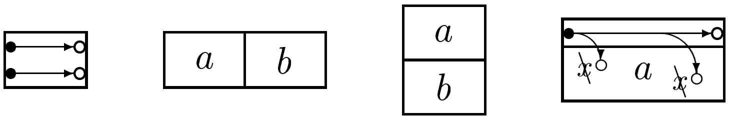
4.4 Definition (Action calculi: statics) A *static action calculus* comprises a signature $\mathcal{K}$, together with the action structure whose actions are the molecular forms over $\mathcal{K}$, and whose operations are defined as follows. Assume $a = (\vec{u})\,\vec{\lambda}\,\langle\vec{v}\rangle$ and $b = (\vec{x})\,\vec{\mu}\,\langle\vec{y}\rangle$ are molecular forms in which no name which is bound in one occurs in the other. Then we define

$$\begin{array}{rcll} \mathbf{id}_m & \stackrel{\text{def}}{=} & (\vec{x})\langle\vec{x}\rangle & (\vec{x} : m) \\ a \cdot b & \stackrel{\text{def}}{=} & (\vec{u})\,\vec{\lambda}\,\sigma\vec{\mu}\,\langle\sigma\vec{y}\rangle & (\sigma = \{\vec{v}/\vec{x}\}) \\ a \otimes b & \stackrel{\text{def}}{=} & (\vec{u}\vec{x})\vec{\lambda}\,\vec{\mu}\,\langle\vec{v}\vec{y}\rangle & \\ \mathbf{ab}_x a & \stackrel{\text{def}}{=} & (x\vec{u})\,\vec{\lambda}\,\langle x\vec{v}\rangle & \end{array}$$

where $\{\vec{v}/\vec{x}\}$ is the simultaneous substitution of $\vec{v}$ for $\vec{x}$.

We call this the static action calculus over $\mathcal{K}$, and denote it by $\mathsf{AC}^s(\mathcal{K})$. ∎

These operations are easily represented graphically. We shall not define the operations on graphs formally, but just suggest them to the reader as follows:



$\mathbf{id}_2$ $\quad a \cdot b \quad$ $a \otimes b$ $\quad \mathbf{ab}_x a$

In constructing the graph of $\mathbf{ab}_x a$, an arc is taken from the new source (shown) to each sink labelled x in a, and the label x is removed (shown as $\cancel{x}$).

It is easy to establish the following, justifying our definition:

4.5 Proposition With the operations of Definition 4.4, $\mathsf{AC}^s(\mathcal{K})$ is a static action structure. ∎

We now introduce the naming actions exactly as for PIC (see 3.6):

4.6 Definition (Naming actions) The *datum* $\langle x \rangle$ (for each name x) and the *discard* $\boldsymbol{\omega}$ are actions defined as follows:

$$\begin{array}{rcl} \langle x \rangle & \stackrel{\text{def}}{=} & (\,)\langle x \rangle \\ \boldsymbol{\omega} & \stackrel{\text{def}}{=} & (x)\langle\,\rangle\ . \end{array}$$

■

We also generalise the control constants of PIC (see 3.7) to control *operations*:

4.7 Definition (Control operations) Each control K, is defined as a *control operation* upon molecular forms as follows:

$$K(\vec{a}) \quad \stackrel{\text{def}}{=} \quad (\vec{x})\,\langle\vec{x}\rangle K\vec{a}(\vec{y})\,\langle\vec{y}\rangle \quad (\vec{x},\vec{y} \text{ not free in } \vec{a})\ .$$

■

It is now easy to establish

4.8 Proposition (Generation) All actions in $\mathsf{AC}^s(\mathcal{K})$ can be defined using data, discard and controls together with the action structure operations. ■

However, $\mathsf{AC}^s(\mathcal{K})$ is not *freely* generated in this way. Certain equations, such as $\langle x \rangle \cdot \boldsymbol{\omega} = \mathbf{id}_\epsilon$, hold in $\mathsf{AC}^s(\mathcal{K})$ but are not provable from the axioms of action structures. This fact motivates our introduction of further axioms in Section 6.

4.9 Derived abstraction The reader may query the definition of abstraction $\mathbf{ab}_x a \stackrel{\text{def}}{=} (x\vec{u})\,\vec{\lambda}\,\langle x\vec{v}\rangle$, for $a = (\vec{u})\,\vec{\lambda}\,\langle\vec{v}\rangle$. Why is x exported? Indeed, if $a : m \to n$ one might expect the arity $\mathbf{ab}_x a : k \otimes m \to n$ rather than $\mathbf{ab}_x a : k \otimes m \to k \otimes n$. In fact we may define another (but non-functorial) form of abstraction as follows:

$$(x)a : k \otimes m \to n \stackrel{\text{def}}{=} (x\vec{u})\,\vec{\pi}\,\langle\vec{v}\rangle\ .$$

Then we can easily show that $(x)a$ and $\mathbf{ab}_x a$ are interexpressible, thus:

$$\begin{array}{rcl} (x)a & = & \mathbf{ab}_x a \cdot (\boldsymbol{\omega} \otimes \mathbf{id}) \\ \mathbf{ab}_x a & = & (x)(\langle x \rangle \otimes a)\ . \end{array}$$

It appears that each form of abstraction has advantages; neither is clearly more convenient for all purposes.

We can think of the free names $\mathbf{fn}\,a$ of a as the names upon which a "depends" non-trivially. Another way of thinking of a "depending" upon x is that $\mathbf{ab}_x a$ should differ from $\mathbf{ab}_y a$ for any y not free in a. This intuition is justified by the following:

4.10 Proposition For every action a and name x, each of the equations $\mathbf{ab}_x a = \mathbf{id} \otimes a$ and $(x)a = \boldsymbol{\omega} \otimes a$ holds if and only if x does not occur free in an action a. ■

Indeed this is suggested by our action graphs; in the graph for $\mathbf{ab}_x a$, if x is not free in a then no arcs are added in addition to the horizontal one.

We now introduce reaction rules, in order to provide the dynamics of molecular forms.

4.11 Definition (Control rule) A *control rule* over a signature $\mathcal{K}$ takes the form

$$t[\vec{a}] \searrow t'[\vec{a}] \ ,$$

where t and t' are terms built from metavariables $\vec{a}$ using data, discard and controls together with the action structure operations. ■

The use of terms t (rather than molecular forms) in presenting control rules anticipates the term algebra introduced in Section 6. The use of metavariables $\vec{a}$ is not strictly necessary, since the above rule amounts simply to the infinite family of rules gained by replacing $\vec{a}$ by terms. But, as our examples show, the use of metavariables often allows a finite presentation of the family.

4.12 Definition (Action calculi: dynamics) A (*dynamic*) *action calculus* comprises a signature $\mathcal{K}$ and a set $\mathcal{R}$ of control rules over $\mathcal{K}$, together with the action structure $\mathsf{AC}^s(\mathcal{K})$ equipped with the smallest reaction relation $\searrow$ which satisfies the rules $\mathcal{R}$ (for all replacements of metavariables $\vec{a}$ by actions).

We call this the (dynamic) action calculus over $\mathcal{K}$ and $\mathcal{R}$, and denote it by $\mathsf{AC}(\mathcal{K}, \mathcal{R})$. ■

Note that $\mathsf{AC}^s(\mathcal{K})$ is essentially $\mathsf{AC}(\mathcal{K},\emptyset)$. When $\mathcal{R}$ is understood, we often write $\mathsf{AC}(\mathcal{K})$ to mean $\mathsf{AC}(\mathcal{K}, \mathcal{R})$.

We have seen that the reaction relation of PIC= $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{in})$ is generated by the control rule

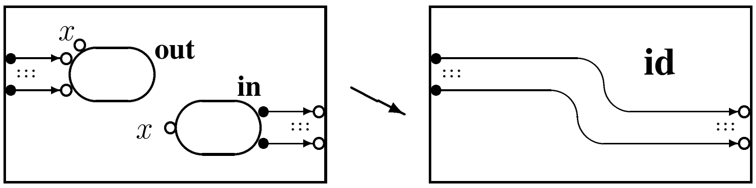
$$\mathbf{out}_x \otimes \mathbf{in}_x \searrow \mathbf{id}_m,$$

where $\mathbf{out}_x = (\langle x \rangle \otimes \mathbf{id})\cdot\mathbf{out}$ and $\mathbf{in}_x = \langle x \rangle\cdot\mathbf{in}$; that is, it is the smallest preorder preserved by the action structure operations which satisfies this rule.

Expressed in molecular form, the control rule of PIC is

$$(\vec{y})\,[\ \langle x\vec{y}\rangle\mathbf{out},\ \langle x\rangle\mathbf{in}(\vec{z})\ ]\,\langle\vec{z}\rangle \searrow (\vec{y})\,\langle\vec{y}\rangle\ .$$

It may also be presented graphically:



The controlling power of controls (not of rank 0) is due to the fact that a reaction relation need not be preserved by controls; e.g. we may have $a \searrow a'$ but not $Ka \searrow Ka'$.

Control mechanisms of arbitrary complexity may be introduced into action calculi; we do not expect to postulate a basic family of controls which is in some sense complete. But the action calculus framework can save work in setting up calculi for interaction, since only the controls need to be specified – and indeed may be shared among the calculi. It also allows us more readily to compare and classify such calculi, on the basis of their controls and associated dynamic rules.

We conclude this section by mentioning briefly the simplest action calculus of all.

4.13 The trivial action calculus The simplest action calculus is $\mathsf{AC}^s(\emptyset)$. It consists just of body-free molecular forms $(\vec{x})[\ \ ]\langle\vec{y}\rangle$, with the identity relation as reaction relation. Proposition 4.8 asserts that it is generated by the naming actions together with the action structure operations. The graphical forms contain no ovals (molecules); they are just "wiring". We may think of $\mathsf{AC}^s(\emptyset)$ as the connective tissue which we use to build more interesting action calculi; we present several examples of these in the following section.

5 Examples of action calculi

We start this section with a review of PIC, which is just $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{in})$. 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 fragments of π-calculi, with additional controls: *boxing* (which may also be called *guarding*), *choice* and *replication*. These all come into π-calculus as defined in [13], and together they appear to provide the same expressive power. We continue with an action calculus for the λ-calculus in 5.5, and conclude the section by outlining an action calculus of Petri nets in 5.6. In most cases we use action graphs to illustrate the control rules.

In one case, the π-calculus with boxing, we state a theorem which asserts that the action calculus represents, in a precise sense, the fragment of the π-calculus to which it corresponds. Similar theorems are believed to hold in the other cases.

Our presentation in each case is quite brief, since (once the arities are defined) each action calculus is determined just by its controls and their dynamics. For PIC and its extensions, we take the arities to be the natural numbers. For the λ-calculus and Petri nets we need a little more structure on the arities.

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

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

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

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

but in fact these two rules generate the same reaction relation, because of the closure conditions. In Section 4.4 we presented this control rule in graphical form.

5.2 π-calculus with boxing: $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$

We now present a more realistic fragment of the π-calculus. In order to emphasize that the representation is precise, we shall present the calculus and its reduction rules first as a process calculus in the style of [13], then as an action calculus. Finally, we state a proposition which asserts that the latter is a faithful representation of the former.

The terms P of the process calculus $\mathcal{P}$ are

$$P \ ::= \ \mathbf{0} \ \mid \ \overline{x}\langle\vec{y}\rangle \ \mid \ x(\vec{y}).P \ \mid \ P|Q \ \mid \ (\boldsymbol{\nu}x)P \ .$$

The first is the empty process; the second is a message $\vec{y}$ sent along channel x; the third is the input along x of a message which is bound to $\vec{y}$ in the continuation P; the fourth is parallel composition; the last is restriction. This variant of π-calculus allows *input* guards $x(\vec{y}).P$, but not *output* guards $\overline{x}\langle\vec{y}\rangle.P$. It was introduced by Honda and Tokoro [8], who show that output guarding can in fact be defined in terms of input guarding.

As in [13], we first define a structural congruence relation $\equiv$ over process terms, by the following equations:

$$\begin{array}{l} P \equiv Q \text{ whenever } P \text{ is alpha-convertible to } Q \ ; \\ P|Q \equiv Q|P, \ \ P|(Q|R) \equiv (P|Q)|R, \ \ P|\mathbf{0} \equiv P \ ; \\ (\boldsymbol{\nu}x)(\boldsymbol{\nu}y)P \equiv (\boldsymbol{\nu}y)(\boldsymbol{\nu}x)P, \ \ (\boldsymbol{\nu}x)(P|Q) \equiv P|(\boldsymbol{\nu}x)Q \ (x \text{ not free in } P) \ . \end{array}$$

Then the reduction relation $\rightarrow$ over terms is the smallest closed under structural congruence which obeys the following rules:

$$\text{COMM}: \ \overline{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \ \rightarrow \ \{\vec{z}/\vec{y}\}P$$

$$\text{PAR}: \ \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad\qquad \text{RES}: \ \frac{P \rightarrow P'}{(\boldsymbol{\nu}x)P \rightarrow (\boldsymbol{\nu}x)P'}$$

Note in particular that from $P \rightarrow P'$ we cannot infer $x(\vec{y}).P \rightarrow x(\vec{y}).P'$; the input prefix *guards* the reduction of P.

We now define the action calculus $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$, in the same style as PIC:

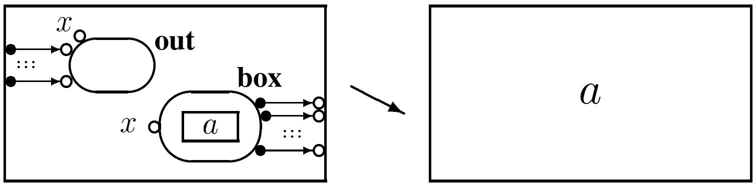
Controls	$\mathbf{out}$	(rank 0)
	$\mathbf{box}$	(rank 1)
Arity rules	$\mathbf{out} \ : 1{+}m \rightarrow 0$	$\dfrac{a \ : m \rightarrow n}{\mathbf{box}\, a \ : 1 \rightarrow n}$
Derived controls	$\mathbf{out}_x \ \stackrel{\text{def}}{=} \ (\langle x\rangle \otimes \mathbf{id}_m)\cdot\mathbf{out}$	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

It is clear that $\mathbf{in} : 0 \rightarrow m$ is redundant in the presence of $\mathbf{box}$, as it is essentially $\mathbf{box}\,\mathbf{id}_m$.

In Section 4.12 we presented the simpler rule $\mathbf{out}_x \otimes \mathbf{in}_x \searrow \mathbf{id}$ both in molecular form and graphically. To show the correspondence, let us do the same for the $\mathbf{box}$ rule. In molecular form it is

$$(\vec{y})\,[\ \langle x\vec{y}\rangle\mathbf{out}, \ \langle x\rangle\mathbf{box}a(\vec{z})\ ]\,\langle\vec{z}\rangle \ \searrow \ a \ ,$$

while in graphical form it is



We now address the question of how the process calculus $\mathcal{P}$ is embedded in $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$. In fact, we shall define a translation $\widehat{(-)} : \mathcal{P} \to \mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$. The translation $\widehat{P}$ will always have arity $0 \to 0$, and is defined as follows (using $\langle \vec{y} \rangle$ to stand for $\langle y_1 \rangle \otimes \cdots \otimes \langle y_r \rangle$):

$$\begin{array}{rcl} \widehat{\mathbf{0}} & \stackrel{\text{def}}{=} & \mathbf{id}_0 \\ \widehat{\overline{x}\langle \vec{y} \rangle} & \stackrel{\text{def}}{=} & \langle x\vec{y} \rangle \cdot \mathbf{out} \\ \widehat{x(\vec{y}).P} & \stackrel{\text{def}}{=} & \langle x \rangle \cdot \mathbf{box}((\vec{y})\widehat{P}) \\ \widehat{P \mid Q} & \stackrel{\text{def}}{=} & \widehat{P} \otimes \widehat{Q} \\ \widehat{(\boldsymbol{\nu} x)P} & \stackrel{\text{def}}{=} & \boldsymbol{\nu} \cdot (x)\widehat{P} \,. \end{array}$$

We should observe that not every action $a : 0 \to 0$ of $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$ lies in the image of this translation. In particular, since $\widehat{P}$ has always arity $0 \to 0$, the action parameter of a **box** molecule always has arity of form $m \to 0$, and hence the **box** molecule binds no names. In fact the action calculus framework allows dataflow between processes in a way in which the standard π-calculus (and in particular the fragment $\mathcal{P}$ considered here) does not.

The following theorem asserts that $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$ represents $\mathcal{P}$ faithfully, via our translation.

Theorem For all P and Q in $\mathcal{P}$:

(1) $P \equiv Q$ iff $\widehat{P} = \widehat{Q}$.
(2) If $P \to Q$ then $\widehat{P} \searrow^1 \widehat{Q}$.
(3) If $\widehat{P} \searrow^1 a$ then for some P', $P \to P'$ and $\widehat{P'} = a$.

■

We omit the proof. It is best done with the help of a (partial) inverse translation from $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box})$ to $\mathcal{P}$; this can be defined inductively on the structure of molecular forms.

5.3 $\boldsymbol{\pi}$-calculus with boxing and choice: $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box}, \mathbf{choose})$

The foregoing π-calculus $\mathcal{P}$ can be refined in several ways, and we shall briefly discuss action calculi corresponding to two of them. In each case there should be no difficulty in proving a theorem which asserts that the correspondence is faithful.

The first refinement is to add choice. We shall restrict consideration to choice among inputs. This amounts to replacing the input form $x(\vec{y}).P$ by a finite sum of such forms; then the COMM rule is changed to

$$\text{COMM}: \ \overline{x}\langle \vec{z} \rangle \mid (\ldots + x(\vec{y}).P + \ldots) \to \{\vec{z}/\vec{y}\}P$$

To match this in an action calculus, we simply add a control **choose** of variable rank and modify the control rule, as follows:

Controls	**out**	(rank 0)
	box	(rank 1)
	choose	(variable rank)
Arity rules	$\mathbf{out} \ : 1{+}m \to 0$	$\dfrac{a \ : m \to n}{\mathbf{box}\, a \ : 1 \to n}$
	$\dfrac{a_1 \ : 0 \to n \qquad \cdots \qquad a_r \ : 0 \to n}{\mathbf{choose}(a_1, \ldots, a_r) \ : 0 \to n}$	
Derived controls	$\mathbf{out}_x \stackrel{\text{def}}{=} (\langle x \rangle \otimes \mathbf{id}_m) \cdot \mathbf{out}$	
	$\mathbf{box}_x a \stackrel{\text{def}}{=} \langle x \rangle \cdot \mathbf{box}\, a$	
Control rule	$\mathbf{out}_x \otimes \mathbf{choose}(\vec{b_1},\ \mathbf{box}_x a,\ \vec{b_2}) \searrow a$	

The reader should have no difficulty in defining controls to allow choice among both inputs and outputs, as in [13].

5.4 π-calculus with boxing and replication: $\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box}, \mathbf{rep})$

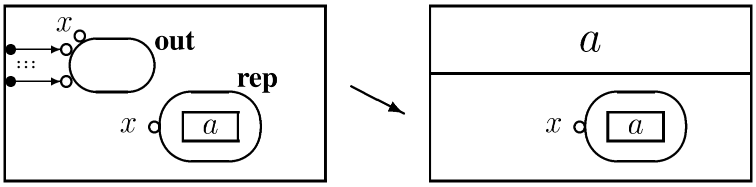
Our second refinement of $\mathcal{P}$ to add replication. We consider a more specific construction than the form $!P$ of [13], but one which covers most current uses of replication. We add a new form $!x(\vec{y}).P$ with reduction rule

$$\text{REP}: \ \overline{x}\langle \vec{z} \rangle \mid !x(\vec{y}).P \ \rightarrow \ \{\vec{z}/\vec{y}\}P \mid !x(\vec{y}).P \ .$$

The matching action calculus is as follows:

Controls	**out**	(rank 0)
	box, **rep**	(rank 1)
Arity rules	$\mathbf{out} \ : 1{+}m \to 0$	$\dfrac{a \ : m \to n}{\mathbf{box}\, a \ : 1 \to n}$
	$\dfrac{a \ : m \to 0}{\mathbf{rep}\, a \ : 1 \to 0}$	
Derived controls	$\mathbf{out}_x \stackrel{\text{def}}{=} (\langle x \rangle \otimes \mathbf{id}_m) \cdot \mathbf{out}$	
	$\mathbf{box}_x a \stackrel{\text{def}}{=} \langle x \rangle \cdot \mathbf{box}\, a$	
	$\mathbf{rep}_x a \stackrel{\text{def}}{=} \langle x \rangle \cdot \mathbf{rep}\, a$	
Control rules	$\mathbf{out}_x \otimes \mathbf{box}_x a \searrow a$	
	$\mathbf{out}_x \otimes \mathbf{rep}_x a \searrow a \otimes \mathbf{rep}_x a$	

The graphical form of the second control rule is



A more explicit form of the rule is

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

which shows clearly that a copy of the replicated action is "spun off" with parameters $\vec{z}$ received via x.

All our extensions of PIC can of course be combined into

$$\mathsf{AC}(\boldsymbol{\nu}, \mathbf{out}, \mathbf{box}, \mathbf{choose}, \mathbf{rep})\;;$$

for practical purposes this appears to be as expressive as the π-calculus as presented in [13]. The original π-calculus [19] contained an operator called *matching*, which is also expressible as a control.

5.5 $\boldsymbol{\lambda}$-calculus: LAMC = AC($\boldsymbol{\lambda}$, **ap**)

The controls in LAMC represent lambda-abstraction and function application. To state their arity rules we need a little more structure upon the arity monoid M, freely generated by a set P of primes. We require exponentiation, in the form of an injective map $\Rightarrow: M \times M \to P$. Thus for each pair m, n of arities there is a prime "functional" arity $m \Rightarrow n$. (Since $\Rightarrow$ is injective, the set P of primes is at least denumerably infinite.)

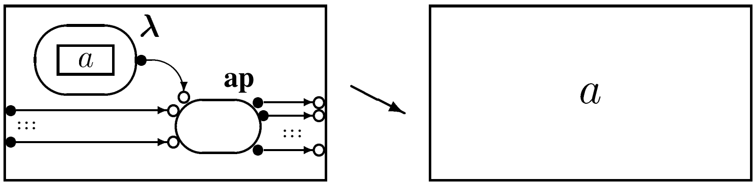
Controls		$\boldsymbol{\lambda}$	(rank 1)
		ap	(rank 0)
Arity rules		$\dfrac{a : m \to n}{\boldsymbol{\lambda} a : \epsilon \to m \Rightarrow n}$	$\mathbf{ap} : (m \Rightarrow n) \otimes m \to n$
Control rules	$\boldsymbol{\sigma}$:	$(\boldsymbol{\lambda} a \otimes \mathbf{id}) \cdot (x)b \searrow \{{}^{\boldsymbol{\lambda} a}\!/\!_x\}b$	$(a : m \to n,\; x : m \Rightarrow n)$
	$\boldsymbol{\beta}$:	$(\boldsymbol{\lambda} a \otimes \mathbf{id}_m) \cdot \mathbf{ap} \searrow a$	$(a : m \to n)$

The $\boldsymbol{\sigma}$ rule distributes the "code" $\boldsymbol{\lambda} a$ to all points where it is used. The substitution $\{{}^{\boldsymbol{\lambda} a}\!/\!_x\}$ means that any free occurrence $\langle x\rangle$ of x in b –and this is the only possible kind of free occurrence– is replaced by $\boldsymbol{\lambda} a$. This rule can in fact be replaced by rules which perform the substitution incrementally, rather than in a single step; this brings LAMC into closer correspondence with the $\lambda\sigma$-calculus of explicit substitutions (Abadi et al [1]).

The $\boldsymbol{\beta}$ rule corresponds to β-reduction. It is revealing to compare it with the **box** rule of Section 4.4. In molecular form it is

$$(\vec{y})\,[\;\boldsymbol{\lambda} a(x),\;\langle x\vec{y}\rangle\mathbf{ap}(\vec{z})\;]\,\langle\vec{z}\rangle \;\searrow\; a\;,$$

while in graphical form it is

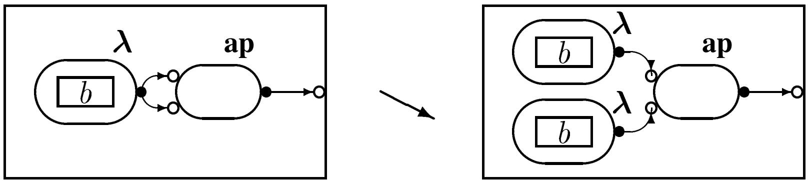


The two controls $\boldsymbol{\lambda}$ and **ap** were introduced in [18] for the more general purpose of lifting an arbitrary action calculus $\mathsf{AC}(\mathcal{K})$ to higher order. (In that paper, the code $\boldsymbol{\lambda}a$ was written $\ulcorner a \urcorner$.) Thus the LAMC as presented here is just the lifting of $\mathsf{AC}(\emptyset)$. This illustrates again the sharing of controls among different action calculi.

Our prescription of the arity monoid $(M, \otimes, \epsilon)$ for LAMC allows some freedom. One alternative is to take the monoid *freely generated* by some basic elements B and the binary operation $\Rightarrow$; in that case, the primes are $P = B \cup \{m \Rightarrow n \mid m, n \in M\}$ and are all distinct, so we have effectively the simply typed λ-calculus. But in general, our requirement that $\Rightarrow : M \times M \to P$ be an injection allows that $p = p \Rightarrow p$ for some prime p. In that case, we have the type-free λ-calculus embedded in LAMC. The embedding is quite easy to define, just as we defined an embedding of (part of) the π-calculus in 5.2; we omit details. As an example, consider the λ-term $\Omega = \omega\omega$ where $\omega = \lambda x.xx$ (self-application). Define **copy** $\stackrel{\text{def}}{=} (x)\langle xx \rangle$ and let $b = \mathbf{copy} \cdot \mathbf{ap}$; then Ω and its infinite reduction are represented by

$$(\boldsymbol{\lambda}b \otimes \boldsymbol{\lambda}b) \cdot \mathbf{ap} \searrow \boldsymbol{\lambda}b \cdot b \searrow (\boldsymbol{\lambda}b \otimes \boldsymbol{\lambda}b) \cdot \mathbf{ap} \searrow \ \dots ,$$

the first two reductions being by the $\boldsymbol{\beta}$ and $\boldsymbol{\sigma}$ rules respectively. The $\boldsymbol{\sigma}$ reduction appears thus in graphical form:



$$\boldsymbol{\lambda}b \cdot \mathbf{copy} \cdot \mathbf{ap} \searrow (\boldsymbol{\lambda}b \otimes \boldsymbol{\lambda}b) \cdot \mathbf{ap}$$

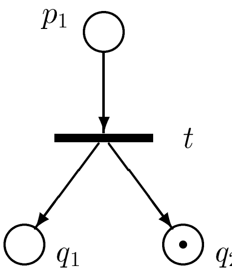
5.6 Petri nets: $\mathsf{NETC} = \mathsf{AC}(\boldsymbol{\nu}, \mathbf{m}, \mathbf{pre}, \mathbf{post})$

We shall present as an action calculus NETC, one of the action structures for Petri nets which were outlined in [15]. It consists first of a monoidal category of place-transition nets in the spirit of Meseguer and Montanari [10]. What is new here is the parametrization of nets both on places and on transitions. There are just two prime arities: P and T. The name-set X is partitioned into two infinite sets; the *place-names* $p, q, \dots : \mathrm{P}$ and the *transition-names* $t, u, \dots : \mathrm{T}$.

The four controls are simple. The first is restriction ($\boldsymbol{\nu}$), in common with PIC. The others correspond to marking a place (**m**), and declaring a place to be either a pre-condition (**pre**) or a post-condition (**post**) of a transition. Here are the constituents of NETC:

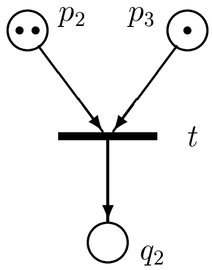
Controls	$\boldsymbol{\nu}$, **m**, **pre**, **post**	(rank 0)
Arity rules	$\boldsymbol{\nu} : \epsilon \to p \;\; (p \in \{\mathrm{P}, \mathrm{T}\})$ $\mathbf{pre} : \mathrm{P} \otimes \mathrm{T} \to \epsilon$	$\mathbf{m} : \mathrm{P} \to \epsilon$ $\mathbf{post} : \mathrm{P} \otimes \mathrm{T} \to \epsilon$
Derived controls	$\mathbf{m}_{\vec{p}} \stackrel{\text{def}}{=} \langle p_1 \rangle \cdot \mathbf{m} \otimes \cdots \otimes \langle p_k \rangle \cdot \mathbf{m}$ $\mathbf{pre}^t_{\vec{p}} \stackrel{\text{def}}{=} \langle p_1 t \rangle \cdot \mathbf{pre} \otimes \cdots \otimes \langle p_k t \rangle \cdot \mathbf{pre}$ $\mathbf{post}^t_{\vec{q}} \stackrel{\text{def}}{=} \langle q_1 t \rangle \cdot \mathbf{post} \otimes \cdots \otimes \langle q_\ell t \rangle \cdot \mathbf{post}$ $\mathbf{trans}_{\vec{p};\vec{q}} \stackrel{\text{def}}{=} \boldsymbol{\nu} \cdot (t)\, (\mathbf{pre}^t_{\vec{p}} \otimes \mathbf{post}^t_{\vec{q}})$	$(\vec{p} : \mathrm{P}^k)$ $(\vec{p} : \mathrm{P}^k)$ $(\vec{q} : \mathrm{P}^\ell)$
Control rule	$\mathbf{m}_{\vec{p}} \otimes \mathbf{trans}_{\vec{p};\vec{q}} \searrow \mathbf{m}_{\vec{q}} \otimes \mathbf{trans}_{\vec{p};\vec{q}}$	

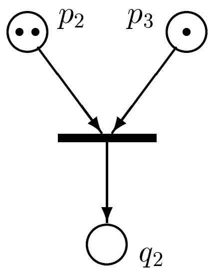
We shall first consider actions of arity $\epsilon \to \epsilon$; they are just place-transition nets in which some or all of the places and transitions bear names, each name occurring at most once in a net. To each arc and each token of a net corresponds a single particle. Here are three examples of Petri nets as they are usually drawn, together with their algebraic forms:



$$a = \mathbf{pre}^t_{p_1} \otimes \mathbf{post}^t_{q_1 q_2} \otimes \mathbf{m}_{q_2}$$

$$b = \mathbf{pre}^t_{p_2 p_3} \otimes \mathbf{post}^t_{q_2} \otimes \mathbf{m}_{p_2 p_2 p_3}$$





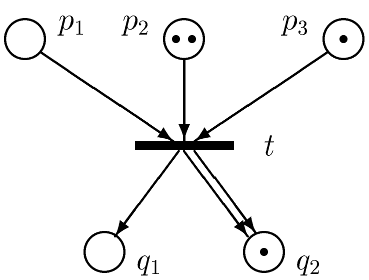
$$c = \boldsymbol{\nu} \cdot (t) b$$

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.

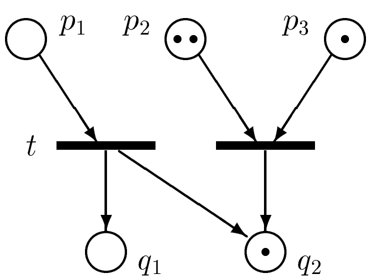
Now compare b with the third net c, whose transition is not named. The name has been hidden by composing restriction, $\boldsymbol{\nu}$, with an abstraction.

This difference between named and un-named transitions is crucial for the algebraic operations upon nets in NETC. In forming the product of two nets, like-named places and transitions are coalesced. Thus the two products $a \otimes b$ and $a \otimes c$ are different:

$a \otimes b$



$a \otimes c$



Let us now use this distinction to explain the control rule – especially the rôle played by $\boldsymbol{\nu}$ in the rule. Consider $a \otimes c$ first. According to the normal understanding of Petri nets, the un-named transition of $a \otimes c$ can fire – as it can in c alone. Indeed, this is allowed by our control rule, since we note that

$$c \;=\; \mathbf{m}_{p_2 p_2 p_3} \otimes \mathbf{trans}_{p_2 p_3 ; q_2} \;,$$

so the rule allows the reaction $c \searrow c'$, where

$$c' = \; \mathbf{m}_{p_2 q_2} \otimes \mathbf{trans}_{p_2 p_3 ; q_2} \;;$$

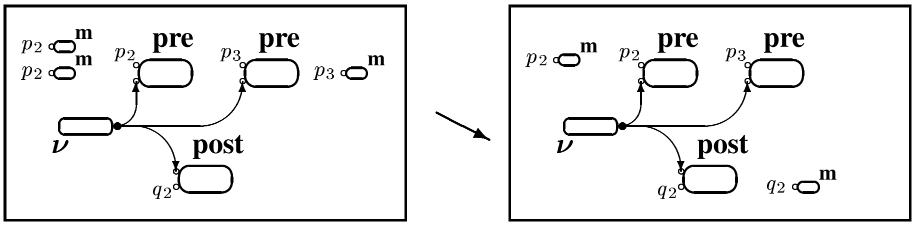
a token has been removed from each of p_2, p_3 and one placed on q_2, exactly as is dictated by the conventional net firing rule.

Now consider $a \otimes b$. According to the normal understanding of Petri nets, its single transition cannot fire (since it lacks a precondition); therefore, because product ($\otimes$) must preserve reaction, the single transition of b must be unable to react in NETC. Indeed, the control rule does not allow it to react, since it contains no restriction.

In NETC, 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 a transition's name is restricted (i.e. removed from the net-diagram) that we can assume its description to be *complete*, and thus know when it may fire.

This shows that NETC contains more than Petri nets as they are normally considered; it contains also partial descriptions of nets. Such partial descriptions can be combined by product and composition; they can also be instantiated, using abstractions. Such instantiation may even coalesce two existing (named) places; for example 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.

Hitherto we have drawn Petri nets in the conventional way. As an example of how they appear as action graphs, here is the reaction $c \searrow c'$ discussed above. Note how a control molecule $\boldsymbol{\nu}$ represents an un-named transition.



$$\mathbf{m}_{p_2 p_2 p_3} \otimes \mathbf{trans}_{p_2 p_3 ; q_2} \;\searrow\; \mathbf{m}_{p_2 q_2} \otimes \mathbf{trans}_{p_2 p_3 ; q_2}$$

We now have two ways to present Petri nets graphically; the conventional way, and as action graphs. It appears that the former is just a streamlined version of the latter, exploiting the particular nature of the NETC controls.

Space prevents us from discussing the constructions of this action calculus further, but several points arise even from what has been said.

First, it appears 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 parametric Petri nets; its algebraic treatment appears remarkably natural, and can be based on four simple controls.

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 the structure of 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.

Fourth, it is striking that only controls with rank 0 are needed to define nets. This suggests that careful study is needed to determine in what sense parametric controls add expressive power to action calculi.

6 The equational theory of action calculi

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

The characterisation is this: For each set $\mathcal{K}$ of controls, the action calculus $\mathsf{AC}(\mathcal{K})$ is isomorphic to the quotient of a term algebra by a certain congruence. The terms are generated by the data, discard, action structure and control operators; the congruence is induced by the basic action structure axioms together with further axioms. The latter, which we shall call the *naming* axioms, are common to all action calculi.

To define the terms we first introduce constants $\langle x\rangle : \epsilon \to p$ (for each $x : p$) and $\boldsymbol{\omega} : p \to \epsilon$ (for each p), corresponding to the naming actions.

6.1 Definition (Terms) The *terms* over $\mathcal{K}$, denoted by $\mathsf{T}(\mathcal{K})$, are generated as follows (we let s, t range over terms):

$$t \ ::= \ \mathbf{id} \ \mid \ s \cdot t \ \mid \ s \otimes t \ \mid \ \mathbf{ab}_x t \ \mid \ \langle x\rangle \ \mid \ \boldsymbol{\omega} \ \mid \ K\vec{t}$$

where the constructions have arities dictated by the arity rules. The notions of *free name* and *bound name* are standard; $\mathbf{ab}_x$ binds x and $\langle x\rangle$ represents a free occurrence of x. The set of names free in t is denoted by $\mathbf{fn}\, t$. ■

6.2 Definition (Derived operations) We define an alternative form $(x)t$ of *abstraction* (see also 4.8, 4.9) and the *permutations* $\mathbf{p}_{mn}$ as follows, together with some abbreviations:

$$\begin{array}{llcll}
(1) & (x)t & \stackrel{\text{def}}{=} & \mathbf{ab}_x t \cdot (\boldsymbol{\omega} \otimes \mathbf{id}) & \\
 & (\vec{x})t & \stackrel{\text{def}}{=} & (x_1)\cdots(x_r)t & (\vec{x} = x_1 \cdots x_r,\ \text{all distinct},\ r \geq 0) \\
 & \langle\vec{x}\rangle & \stackrel{\text{def}}{=} & \langle x_1\rangle \otimes \cdots \otimes \langle x_r\rangle & (\vec{x} = x_1 \cdots x_r,\ r \geq 0) \\
(2) & \mathbf{p}_{mn} & \stackrel{\text{def}}{=} & (\vec{x}\vec{y})\langle\vec{y}\vec{x}\rangle & (\vec{x} : m,\ \vec{y} : n)\ .
\end{array}$$

■

Note that $\mathbf{p}_{mn}$ is defined using a *particular* vector $\vec{x}\vec{y}$ of distinct variables; when α-conversion is proven, we shall be justified in choosing these variables at will.

6.3 Definition (Theory AC) The *equational theory* AC is the set of equations upon terms generated by the action structure axioms together with the following *naming axioms*:

$$\begin{array}{rll}
\boldsymbol{\gamma}: & (x)t = \boldsymbol{\omega} \otimes t & (x \notin \mathbf{fn}\, t) \\
\boldsymbol{\delta}: & (x)(\langle x\rangle \otimes \mathbf{id}_m) = \mathbf{id}_{p\otimes m} & (x : p) \\
\boldsymbol{\zeta}: & \mathbf{p}_{km} \cdot (t \otimes s) = (s \otimes t) \cdot \mathbf{p}_{\ell n} & (s : k \to \ell,\ t : m \to n) \\
\boldsymbol{\sigma}: & (\langle y\rangle \otimes \mathbf{id}_m) \cdot (x)t = \{y/x\}t & (t : m \to n)\ .
\end{array}$$

■

An equivalent form of $\boldsymbol{\gamma}$ is $\mathbf{ab}_x t = \mathbf{id} \otimes t$; as we saw in Proposition 4.10, this is one way of asserting that x does not appear free in the molecular form corresponding to t. The axiom $\boldsymbol{\delta}$ asserts essentially that abstracting a name from the corresponding datum is just the identity. The axiom $\boldsymbol{\zeta}$ is one of the coherence laws which assert that the permutations $\mathbf{p}$ constitute a *symmetry* on the underlying monoidal category; the other two coherence conditions are provable in AC. Finally, $\boldsymbol{\sigma}$ asserts that to compose a datum with an abstraction is the same as doing a textual substitution.

It is easy to see, informally, that these axioms hold when we think of t as a molecular form. The essence of what follows is that these simple truths are *all* we need to assert, above the action structure axioms, to characterize molecular forms completely.

We shall write $\mathsf{T}(\mathcal{K})/\mathsf{AC}$ to mean the quotient of the term algebra $\mathsf{T}(\mathcal{K})$ by the congruence induced by the theory AC. In order to prove that this quotient is isomorphic to the static action calculus $\mathsf{AC}^s(\mathcal{K})$ (presented in molecular form) we first prove several consequences of the theory AC. We shall write $\mathsf{AC} \vdash s = t$ to mean that $s = t$ is provable in AC.

6.4 Proposition The following are provable in AC:

$$\begin{array}{rlll}
 & (1) & \langle x\rangle \cdot \boldsymbol{\omega} = \mathbf{id}_\epsilon & \\
 & (2) & (x)(s \cdot t) = (x)s \cdot t & (x \notin \mathbf{fn}\, t) \\
\boldsymbol{\alpha}: & (3) & (x)t = (y)\{y/x\}t & (y \notin \mathbf{fn}\, t) \\
 & (4) & (x)(t \otimes \mathbf{id}) = (x)t \otimes \mathbf{id} & \\
 & (5) & (x)(s \otimes t) = (x)s \otimes t & (x \notin \mathbf{fn}\, t) \\
 & (6) & \mathbf{ab}_x t = (x)(\langle x\rangle \otimes t) & \\
 & (7) & \mathbf{ab}_x t = \mathbf{id} \otimes t & (x \notin \mathbf{fn}\, t) \\
 & (8) & (\vec{x}\vec{y})t = (\mathbf{p}_{mn} \otimes \mathbf{id}) \cdot (\vec{y}\vec{x})t & (\vec{x} : m, \vec{y} : n) \\
 & (9) & (\vec{x})\langle\vec{x}\rangle = \mathbf{id}\ . &
\end{array}$$

Proof

(1)
$$\begin{array}{rcll}
\mathsf{AC} \vdash \langle x\rangle \cdot \boldsymbol{\omega} & = & \langle x\rangle \cdot (x)\mathbf{id}_\epsilon & \boldsymbol{\gamma} \\
 & = & \mathbf{id}_\epsilon\ . & \boldsymbol{\sigma}
\end{array}$$

(2)
$$\begin{array}{rcll}
\mathsf{AC} \vdash (x)(s \cdot t) & = & \mathbf{ab}_x s \cdot (x)t & 6.2(1) \\
 & = & \mathbf{ab}_x s \cdot (\boldsymbol{\omega} \otimes t) & \boldsymbol{\gamma} \\
 & = & \mathbf{ab}_x s \cdot (\boldsymbol{\omega} \otimes \mathbf{id}) \cdot t & \\
 & = & (x)s \cdot t\ . & 6.2(1)
\end{array}$$

$$
\begin{array}{llcll}
(3) & \mathsf{AC} \vdash (y)\{y/x\}t & = & (y)\big((\langle y\rangle \otimes \mathbf{id}) \cdot (x)t\big) & \boldsymbol{\sigma} \\
 & & = & (y)(\langle y\rangle \otimes \mathbf{id}) \cdot (x)t & (2) \\
 & & = & (x)t\ . & \boldsymbol{\delta}
\end{array}
$$

$$
\begin{array}{llcll}
(4) & \mathsf{AC} \vdash (x)(t \otimes \mathbf{id}) & = & (x)(\{x/x\}t \otimes \mathbf{id}) & \\
 & & = & (x)\big((\langle x\rangle \otimes \mathbf{id}) \cdot (x)t \otimes \mathbf{id}\big) & \boldsymbol{\sigma} \\
 & & = & (x)\big((\langle x\rangle \otimes \mathbf{id} \otimes \mathbf{id}) \cdot ((x)t \otimes \mathbf{id})\big) & \\
 & & = & (x)(\langle x\rangle \otimes \mathbf{id} \otimes \mathbf{id}) \cdot \big((x)t \otimes \mathbf{id}\big) & (2) \\
 & & = & (x)t \otimes \mathbf{id}\ . & \boldsymbol{\delta}
\end{array}
$$

$$
\begin{array}{llcll}
(5) & \mathsf{AC} \vdash (x)(s \otimes t) & = & (x)\big((s \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes t)\big) & \\
 & & = & (x)(s \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes t) & (2) \\
 & & = & \big((x)s \otimes \mathbf{id}\big) \cdot (\mathbf{id} \otimes t) & (4) \\
 & & = & (x)s \otimes t\ . &
\end{array}
$$

$$
\begin{array}{llcll}
(6) & \mathsf{AC} \vdash \mathbf{ab}_x t & = & \mathbf{ab}_x t \cdot (x)(\langle x\rangle \otimes \mathbf{id}) & \boldsymbol{\delta} \\
 & & = & (x)\big(t \cdot (\langle x\rangle \otimes \mathbf{id})\big) & 6.2(1) \\
 & & = & (x)(\langle x\rangle \otimes t)\ . &
\end{array}
$$

$$
\begin{array}{llcll}
(7) & \mathsf{AC} \vdash \mathbf{ab}_x t & = & (x)(\langle x\rangle \otimes t) & (6) \\
 & & = & (x)\big((\langle x\rangle \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes t)\big) & \\
 & & = & (x)(\langle x\rangle \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes t) & (2) \\
 & & = & \mathbf{id} \otimes t\ . & \boldsymbol{\delta}
\end{array}
$$

$$
\begin{array}{lcll}
(8)\ \mathsf{AC} \vdash (\mathbf{p}_{mn} \otimes \mathbf{id}) \cdot (\vec{y}\vec{x})t & = & (\vec{x}\vec{y})(\langle \vec{y}\vec{x}\rangle \otimes \mathbf{id}) \cdot (\vec{y}\vec{x})t & 6.2(2), (5)^* \\
 & = & (\vec{x}\vec{y})\big((\langle \vec{y}\vec{x}\rangle \otimes \mathbf{id}) \cdot (\vec{y}\vec{x})t\big) & (2)^* \\
 & = & (\vec{x}\vec{y})t\ . & \boldsymbol{\sigma}^*
\end{array}
$$

(9) Induction on length of $\vec{x}$. Basis true by definition. Step:

$$
\begin{array}{lcll}
\mathsf{AC} \vdash (x\vec{y})\langle x\vec{y}\rangle & = & (x)(\vec{y})\big(\langle \vec{y}\rangle \cdot (\langle x\rangle \otimes \mathbf{id})\big) & \\
 & = & (x)\big((\vec{y})\langle \vec{y}\rangle \cdot (\langle x\rangle \otimes \mathbf{id})\big) & (2) \\
 & = & (x)(\langle x\rangle \otimes \mathbf{id}) & \text{induction} \\
 & = & \mathbf{id}\ . & \boldsymbol{\delta}
\end{array}
$$

■

To demonstrate our isomorphism, we define translations back and forth between $\mathsf{AC}^s(\mathcal{K})$ and $\mathsf{T}(\mathcal{K})/\mathsf{AC}$, and prove that they are inverse to one another.

We begin with a translation $\widehat{(-)} : \mathsf{AC}^s(\mathcal{K}) \to \mathsf{T}(\mathcal{K})/\mathsf{AC}$. The translation $\widehat{a}$ is defined inductively on the structure of the molecular form a. First we translate each $a \in \mathsf{AC}^s(\mathcal{K})$ to a term without assuming that a is subject to alpha-conversion and commutation of molecules. Then we show that, if a can be transformed to b by these operations, their translations are provably equal in AC. The translation $\widehat{(-)}$ is therefore well-defined from $\mathsf{AC}^s(\mathcal{K})$ to $\mathsf{T}(\mathcal{K})/\mathsf{AC}$.

Note that some notations –e.g. $\langle x\rangle$– are used both within molecular forms and for terms. This abuse of notation is always consistent with our translation; it sometimes makes the translation look like an identity function.

6.5 Definition (Translating molecular forms to terms) The function $\widehat{(-)}$ is defined inductively as follows:

$$\widehat{(\vec{x})\,\vec{\mu}\,\langle\vec{y}\rangle} \stackrel{\text{def}}{=} \begin{cases} (\vec{x})\langle\vec{y}\rangle & \text{if } \vec{\mu} \text{ is empty} \\ (\vec{x})(\langle\vec{u}\rangle \cdot K\widehat{\vec{c}} \cdot \widehat{(\vec{x'})\vec{\mu'}\langle\vec{y}\rangle}) & \text{if } \vec{\mu} = \lambda\vec{\mu'}, \text{ where } \lambda = \langle\vec{u}\rangle K\vec{c}(\vec{x'}). \end{cases}$$

■

The following is easily proved by induction on a:

6.6 Proposition $\mathbf{fn}\,\widehat{a} = \mathbf{fn}\,a$. ■

Using this, the following can be proved simultaneously by induction on a:

6.7 Proposition The translation $\widehat{(-)}$ preserves alpha-convertibility and substitution, i.e.

(1) If b is an alpha-variant of a then $\boldsymbol{\alpha} \vdash \widehat{b} = \widehat{a}$;

(2) If σ is a substitution $\{\vec{y}/\vec{x}\}$ then $\boldsymbol{\alpha} \vdash \widehat{\sigma a} = \sigma\widehat{a}$.

■

Now we complete the proof that

6.8 Proposition The translation $\widehat{(-)} : \mathsf{AC}^s(\mathcal{K}) \to \mathsf{T}(\mathcal{K})/\mathsf{AC}$ is well-defined.

Proof We have already shown that the translations of alpha-variant molecular forms are provably equal terms (by $\boldsymbol{\alpha}$). It remains to show that the translations of forms which only differ by admissible commutation of molecules are provably equal in AC.

It will be enough to show for $a : \epsilon \to n$ that

$$\text{if } a_1 = \mu_1\,\mu_2\,a \text{ and } a_2 = \mu_2\,\mu_1\,a \text{ then } \mathsf{AC} \vdash \widehat{a_1} = \widehat{a_2}\ ,$$

provided μ_1 and μ_2 are commutable. So let

$$\mu_1 = \langle\vec{u_1}\rangle K_1\vec{c_1}(\vec{v_1}) \text{ and } \mu_2 = \langle\vec{u_2}\rangle K_2\vec{c_2}(\vec{v_2})\ ,$$

where $\vec{v_1}$ are not free in μ_2 and $\vec{v_2}$ are not free in μ_1. Then

$$\begin{array}{llll} \mathsf{AC} \vdash & \widehat{a_1} & = \langle\vec{u_1}\rangle \cdot K_1\widehat{\vec{c_1}} \cdot (\vec{v_1})(\langle\vec{u_2}\rangle \cdot K_2\widehat{\vec{c_2}} \cdot (\vec{v_2})\widehat{a}) & \\ & & = \langle\vec{u_1}\rangle \cdot K_1\widehat{\vec{c_1}} \cdot \mathbf{ab}_{\vec{v_1}}(\langle\vec{u_2}\rangle \cdot K_2\widehat{\vec{c_2}}) \cdot (\vec{v_1}\vec{v_2})\widehat{a} & \\ & & = \langle\vec{u_1}\rangle \cdot K_1\widehat{\vec{c_1}} \cdot (\mathbf{id} \otimes \langle\vec{u_2}\rangle\cdot K_2\widehat{\vec{c_2}}) \cdot (\vec{v_1}\vec{v_2})\widehat{a} & \text{by 6.4(7)} \\ & & = (\langle\vec{u_1}\rangle\cdot K_1\widehat{\vec{c_1}} \otimes \langle\vec{u_2}\rangle\cdot K_2\widehat{\vec{c_2}}) \cdot (\vec{v_1}\vec{v_2})\widehat{a} & \\ & & = (\langle\vec{u_2}\rangle\cdot K_2\widehat{\vec{c_2}} \otimes \langle\vec{u_1}\rangle\cdot K_1\widehat{\vec{c_1}}) \cdot \mathbf{p}_{m_2 m_1} \cdot (\vec{v_1}\vec{v_2})\widehat{a} & \text{by } \boldsymbol{\zeta} \\ & & = (\langle\vec{u_2}\rangle\cdot K_2\widehat{\vec{c_2}} \otimes \langle\vec{u_1}\rangle\cdot K_1\widehat{\vec{c_1}}) \cdot (\vec{v_2}\vec{v_1})\widehat{a} & \text{by 6.4(8)} \\ & & = \widehat{a_2} & \text{by symmetry.} \end{array}$$

■

It is worth noting that we have used all four axioms $\boldsymbol{\gamma}$, $\boldsymbol{\delta}$, $\boldsymbol{\zeta}$ and $\boldsymbol{\sigma}$ in establishing this well-definedness.

When working in AC we shall often abbreviate $\mathsf{AC} \vdash t = u$ to $t = u$, giving reasons (e.g. the axiom used) where necessary.

From now on we shall write a superscript ${}^{\mathcal{M}}$ thus

$$\mathbf{id}^{\mathcal{M}} \qquad \cdot^{\mathcal{M}} \qquad \otimes^{\mathcal{M}} \qquad \mathbf{ab}_x^{\mathcal{M}} \qquad \langle x\rangle^{\mathcal{M}} \qquad \boldsymbol{\omega}^{\mathcal{M}} \qquad K^{\mathcal{M}}$$

on the seven operations defined over molecular forms in Definitions 4.4, 4.6 and 4.7, to distinguish them from the corresponding term-building operations in $\mathsf{T}(\mathcal{K})$. Our next task is to show that, as we expect, our translation respects this correspondence.

6.9 Proposition The translation $\widehat{(-)}$ preserves the action structure operations, data, discard and controls, i.e. the following are provable in AC:

(1) $\widehat{\mathbf{id}^{\mathcal{M}}} = \mathbf{id}$	(5) $\widehat{\langle x\rangle^{\mathcal{M}}} = \langle x\rangle$
(2) $\widehat{a \cdot^{\mathcal{M}} b} = \widehat{a} \cdot \widehat{b}$	(6) $\widehat{\boldsymbol{\omega}^{\mathcal{M}}} = \boldsymbol{\omega}$
(3) $\widehat{a \otimes^{\mathcal{M}} b} = \widehat{a} \otimes \widehat{b}$	(7) $\widehat{K^{\mathcal{M}}(\vec{c})} = K\widehat{\vec{c}}\,.$
(4) $\widehat{\mathbf{ab}_x^{\mathcal{M}} a} = \mathbf{ab}_x \widehat{a}$	

Proof

(1) By 6.4(9).

(2) Let $a = (\vec{u})\,\vec{\lambda}\,\langle\vec{x}\rangle$ and $b = (\vec{v})\,\vec{\mu}\,\langle\vec{y}\rangle$, where (after suitable alpha-conversion) no name bound in one occurs in the other. We use induction on the length of $\vec{\lambda}$. If $\vec{\lambda}$ is empty then $a = (\vec{u})\langle\vec{x}\rangle$ and $\widehat{a} = (\vec{u})\langle\vec{x}\rangle$. Also $\widehat{b} = (\vec{v})\widehat{b_0}$ where $b_0 = \vec{\mu}\,\langle\vec{y}\rangle$. Now $a \cdot^{\mathcal{M}} b = (\vec{u})\,\sigma\vec{\mu}\,\langle\sigma\vec{y}\rangle$, where $\sigma = \{\vec{x}/\vec{v}\}$; so $\widehat{a \cdot^{\mathcal{M}} b} = (\vec{u})\,\widehat{\sigma b_0} = (\vec{u})\,\sigma\widehat{b_0}$ by 6.7(2). On the other hand

$$\begin{aligned} \widehat{a} \cdot \widehat{b} &= (\vec{u})\langle\vec{x}\rangle \cdot (\vec{v})\widehat{b_0} \\ &= (\vec{u})(\langle\vec{x}\rangle \cdot (\vec{v})\widehat{b_0}) && \text{by 6.4(2)} \\ &= (\vec{u})\,\sigma\widehat{b_0} && \text{by } \boldsymbol{\sigma}\,. \end{aligned}$$

If $\vec{\lambda} = \lambda\vec{\lambda'}$, where $\lambda = \langle\vec{w}\rangle K\vec{c}\,(\vec{u'})$, then let $a' = (\vec{u'})\,\vec{\lambda'}\,\langle\vec{x}\rangle$. Then

$$\begin{aligned} \widehat{a \cdot^{\mathcal{M}} b} &= (\vec{u})(\langle\vec{w}\rangle \cdot K\widehat{\vec{c}} \cdot \widehat{a' \cdot^{\mathcal{M}} b}) && \text{by definition of } \widehat{(-)} \\ &= (\vec{u})(\langle\vec{w}\rangle \cdot K\widehat{\vec{c}} \cdot \widehat{a'} \cdot \widehat{b}) && \text{by induction} \\ &= (\vec{u})(\langle\vec{w}\rangle \cdot K\widehat{\vec{c}} \cdot \widehat{a'}) \cdot \widehat{b} && \text{by 6.4(2)} \\ &= \widehat{a} \cdot \widehat{b}\,. \end{aligned}$$

(3) It is an easy induction to prove that $\widehat{a \otimes^{\mathcal{M}} \mathbf{id}^{\mathcal{M}}} = \widehat{a} \otimes \mathbf{id}$, and that $\widehat{\mathbf{id}^{\mathcal{M}} \otimes^{\mathcal{M}} b} = \mathbf{id} \otimes \widehat{b}$. Then we can use case (2) above to deduce $\widehat{a \otimes^{\mathcal{M}} b} = (\widehat{a} \otimes \mathbf{id}) \cdot (\mathbf{id} \otimes \widehat{b}) = \widehat{a} \otimes \widehat{b}\,.$

(4) From 4.9 we have that $\mathbf{ab}_x^{\mathcal{M}} a = (x)^{\mathcal{M}}(\langle x\rangle^{\mathcal{M}} \otimes^{\mathcal{M}} a)$. So

$$
\begin{array}{rcll}
\widehat{\mathbf{ab}_x^{\mathcal{M}} a} & = & (x)\big(\widehat{\langle x\rangle^{\mathcal{M}} \otimes^{\mathcal{M}} a}\big) & \text{by definition} \\
& = & (x)\big(\widehat{\langle x\rangle^{\mathcal{M}}} \otimes \widehat{a}\big) & \text{by case (3)} \\
& = & (x)\big(\langle x\rangle \otimes \widehat{a}\big) & \text{by (5)} \\
& = & \mathbf{ab}_x \widehat{a} & \text{by 6.4(6).}
\end{array}
$$

(5), (6), (7) Immediate. ■

We now turn to the translation in the other direction, from $\mathsf{T}(\mathcal{K})/\mathsf{AC}$ to $\mathsf{AC}^s(\mathcal{K})$. We first define the obvious translation from $\mathsf{T}(\mathcal{K})$:

6.10 Definition (Translating terms to molecular forms) Define $[\![-]\!] : \mathsf{T}(\mathcal{K}) \to \mathsf{AC}^s(\mathcal{K})$ inductively as follows:

$$
\begin{array}{rclcrcl}
[\![\mathbf{id}]\!] & \stackrel{\text{def}}{=} & (x)\langle x\rangle & \qquad & [\![\langle x\rangle]\!] & \stackrel{\text{def}}{=} & \langle x\rangle^{\mathcal{M}} \\
[\![s \cdot t]\!] & \stackrel{\text{def}}{=} & [\![s]\!] \cdot^{\mathcal{M}} [\![t]\!] & & [\![\boldsymbol{\omega}]\!] & \stackrel{\text{def}}{=} & \boldsymbol{\omega}^{\mathcal{M}} \\
[\![s \otimes t]\!] & \stackrel{\text{def}}{=} & [\![s]\!] \otimes^{\mathcal{M}} [\![t]\!] & & [\![K(\vec{t})]\!] & \stackrel{\text{def}}{=} & K^{\mathcal{M}}[\![\vec{t}]\!] \,. \\
[\![\mathbf{ab}_x t]\!] & \stackrel{\text{def}}{=} & \mathbf{ab}_x^{\mathcal{M}}[\![t]\!] & & & &
\end{array}
$$

■

The following is an easy induction:

6.11 Proposition $\mathbf{fn}\,[\![t]\!] \subseteq \mathbf{fn}\,t$. ■

We cannot expect equality here; note that for terms we have $\mathbf{fn}\,(s \cdot t) = \mathbf{fn}\,s \cup \mathbf{fn}\,t$, but in general for molecular forms $\mathbf{fn}\,(a \cdot b) \neq \mathbf{fn}\,a \cup \mathbf{fn}\,b$; consider $a = \langle x\rangle^{\mathcal{M}}$, $b = \boldsymbol{\omega}^{\mathcal{M}}$.

The following is also an easy induction, since $[\![-]\!]$ is defined inductively on term-structure:

6.12 Proposition The translation $[\![-]\!]$ preserves the action structure operations, data, discard, controls and substitution. ■

We now come to the important property of $[\![-]\!]$:

6.13 Proposition The translation $[\![-]\!]$ is a well-defined function from $\mathsf{T}(\mathcal{K})/\mathsf{AC}$ to $\mathsf{AC}^s(\mathcal{K})$; that is, if $\mathsf{AC} \vdash s = t$ then $[\![s]\!] = [\![t]\!]$.

Proof It is enough to prove the result when $\vdash\ s\ =\ t$ is an instance of a basic axiom of action structures or one of the naming axioms. In each case, one uses the fact that $[\![-]\!]$ is defined inductively on the structure of terms. In the case of $\boldsymbol{\delta}$, it therefore remains to show that $(x)^{\mathcal{M}}(\langle x\rangle^{\mathcal{M}} \otimes^{\mathcal{M}} \mathbf{id}_m^{\mathcal{M}}) = \mathbf{id}_{p\otimes m}^{\mathcal{M}}$; this is routine calculation with the operations as defined in 4.4 and 4.6. The case $\boldsymbol{\zeta}$ is similar. In the case of $\boldsymbol{\sigma}$, one also needs $[\![\{y/x\}t]\!] = \{y/x\}[\![t]\!]$ (Proposition 6.12). The case for $\gamma \vdash (x)t = \boldsymbol{\omega} \otimes t$ $(x \notin \mathbf{fn}\,t)$ works just because $\mathbf{fn}\,[\![t]\!] \subseteq \mathbf{fn}\,t$ (Proposition 6.11). For since also $x \notin \mathbf{fn}\,[\![t]\!]$, we have

$$
\begin{array}{rcll}
[\![(x)t]\!] & = & (x)^{\mathcal{M}}[\![t]\!] & \text{by definition} \\
& = & \boldsymbol{\omega}^{\mathcal{M}} \otimes^{\mathcal{M}} [\![t]\!] & \text{by Props 4.8,6.11} \\
& = & [\![\boldsymbol{\omega} \otimes t]\!] & \text{by definition.}
\end{array}
$$

■

We are now ready to prove that the molecular forms are, effectively, normal forms for the theory AC. We have already established that the two translations $\widehat{(-)}$ and $[\![-]\!]$ are well-defined between $\mathsf{AC}^s(\mathcal{K})$ and $\mathsf{T}(\mathcal{K})/\mathsf{AC}$. We next show that they are mutually inverse:

6.14 Proposition For all terms t, $\mathsf{AC} \vdash \widehat{[\![t]\!]} = t$.

Proof Straightforward, since $[\![-]\!]$ is defined inductively via the operations, and $\widehat{(-)}$ preserves all of them (Proposition 6.9). ■

6.15 Proposition For all molecular forms a, $[\![\widehat{a}]\!] = a$.

Proof We proceed inductively on the size of a. The base case is easy. Now suppose $a = (\vec{x})\,\lambda\vec{\lambda'}\,\langle\vec{y}\rangle$. If $\lambda = \langle\vec{z}\rangle K\vec{c}(\vec{x'})$ then

$$
\widehat{a} \;=\; (\vec{x})(\langle\vec{z}\rangle \cdot K\widehat{\vec{c}} \cdot \widehat{a'}) \qquad \text{where } a' = (\vec{x'})\,\vec{\lambda'}\,\langle\vec{y}\rangle\ .
$$

So we have

$$
\begin{array}{rcll}
[\![\widehat{a}]\!] & = & (\vec{x})^{\mathcal{M}}(\langle\vec{z}\rangle^{\mathcal{M}} \cdot^{\mathcal{M}} K^{\mathcal{M}}[\![\,\widehat{\vec{c}}\,]\!] \cdot^{\mathcal{M}} [\![\widehat{a'}]\!]) & \\
& = & (\vec{x})^{\mathcal{M}}(\langle\vec{z}\rangle^{\mathcal{M}} \cdot^{\mathcal{M}} K^{\mathcal{M}}\vec{c} \cdot^{\mathcal{M}} a') & \text{by induction} \\
& = & a & \text{by calculation.}
\end{array}
$$

■

Finally, with Propositions 6.7–6.9 and 6.12–6.15 we have established

6.16 Theorem (Molecular normal form) The translations $\widehat{(-)}$ and $[\![-]\!]$ between $\mathsf{AC}^s(\mathcal{K})$ and $\mathsf{T}(\mathcal{K})/\mathsf{AC}$ constitute a static isomorphism of action structures, which moreover preserves the control operations $K \in \mathcal{K}$, the data $\langle x\rangle$, the discard $\boldsymbol{\omega}$ and the substitutions $\{\vec{y}/\vec{x}\}$. ■

This static isomorphism may be extended to a dynamic one by defining a reaction relation $\searrow$ either on $\mathsf{AC}^s(\mathcal{K})$ or on $\mathsf{T}(\mathcal{K})$ by control rules, and transferring it to the other by use of the static isomorphism.

7 Related and future work

Since action calculi are a special class of action structures of an operational character, it is important to find the abstract interpretations of each calculus as a particular class of action structures. Indeed, this was a prime reason for inventing the action structure framework. The first step in this direction is taken by Mifsud, Milner and Power [12]. Given a signature $\mathcal{K}$ and set $\mathcal{R}$ of control rules over $\mathcal{K}$, a category $\mathsf{CS}(\mathcal{K},\mathcal{R})$ of action structures with added structure is defined called the *control structures* over $\mathcal{K}$ and $\mathcal{R}$; the action calculus $\mathsf{AC}(\mathcal{K},\mathcal{R})$ (equipped also with the naming and control operations of 4.6 and 4.7) is shown to be initial in CS($\mathcal{K}$,$\mathcal{R}$). These categories are characterized by equational axioms which depend only mildly upon $\mathcal{K}$.

Action calculi rely strongly on the concept of *naming*, since this is the means of building molecular structures, and of defining those configurations which are susceptible to reaction (redexes). Although explicit names $x, y, \ldots$ are handy for this purpose, naming structure can in fact be presented more abstractly. Gardner [5] has defined a class of *closed* (i.e. name-free) action calculi, and demonstrates a precise correspondence with those presented here. Hermida and Power [6] have given a name-free treatment of control structures; their *fibrational control structures* provide deeper mathematical insight into the nature of the interpretations of action calculi.

Since there is a highly developed model theory for the λ-calculus, one expects existing models to take their place naturally as control structures. It remains to verify that this is so, or to tackle any problems which prevent it being so. For process calculi there is not such a well-developed model theory, but the same task must be attempted for models which do exist. In particular, CSP and its failures model should be examined in this framework. Another approach to modelling process calculi has been to quotient the syntactic algebra by a congruence, the most common being bisimilarity. Some work in this direction is being done by Mifsud [11] for a version of the π-calculus. A more general problem is to find a notion of bisimilarity which applies uniformly to all – or a large class of – action calculi. This appears to require some constraint upon the form which control rules may take.

This raises a broader question: Can we find means of classifying action calculi in terms of their control rules? Once we have set up the action calculus framework, it is evident that the entire difference between one action calculus and another lies in their dynamics; we have thus provided a setting in which the variety of dynamic disciplines can be analysed.

References

[1] Abadi, M., Cardelli, L., Curien, P.-L. and Lévy, J.-J., *Explicit substitutions*, ACM POPL Conference, 1990.

[2] Barendregt, H., **The Lambda Calculus**, North Holland, 2nd edition, 1984.

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

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

[5] Gardner, P., *A name-free account of action calculi*, Proc. 11th Conference on Mathematical Foundations of Programming Semantics, Tulane, 1995.

[6] Hermida, C.A. and Power, A.J., *Fibrational control structures*, to appear in Proc. CONCUR, 1995.

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

[8] 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.

[9] Jensen, Ole, Forthcoming PhD thesis, University of Cambridge.

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

[11] Mifsud, A., Forthcoming PhD thesis, University of Edinburgh.

[12] Mifsud, A., Milner, R, and Power, J., *Control structures*, Proceedings of Conference on Logics in Computer Science, 1995.

[13] Milner, R., *The polyadic π-calculus: a tutorial*, in **Logic and Algebra of Specification**, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993, pp203–246.

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

[15] Milner, R., Action structures and the π-calculus, in **Proof and Computation**, ed. H. Schwichtenberg, Series F: Computer and Systems Sciences, NATO Advanced Study Institute, (Proceedings of International Summer School held in Marktoberdorf, Germany, 1993), Springer Verlag 1994, pp219–280.

[16] 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.

[17] Milner, R., *Action calculi, or concrete action structures*, Proc. MFCS Conference, Gdansk, Poland, Lecture Notes in Computer Science, Vol 711, Springer-Verlag, pp105–121, 1993.

[18] Milner, R., *Higher-order action calculi*, Proc. Computer Science Logic 1992, ed. Karl Meinke, Lecture Notes in Computer Science, Vol 832, Springer-Verlag, pp238–260, 1994.

[19] 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.

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