ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**D Duke, University of York, UK and A S Evans, University of Bradford, UK
(Eds)**

BCS-FACS Northern Formal Methods Workshop

Proceedings of the BCS-FACS Northern Formal Methods Workshop,
Ilkley, UK. 23-24 September 1996

Towards a New Conceptual Framework for the Modelling of Dynamically Distributed Systems

M. Mac an Airchinnigh

Springer

Published in collaboration with the
British Computer Society

Towards a New Conceptual Framework for the Modelling of Dynamically Distributed Systems

Mícheál Mac an Airchinnigh
University of Dublin,
Trinity College, Dublin, Ireland
e-mail: {mmaa@cs.tcd.ie, mmaa@kmttech.ie}

Abstract

We hypothesize that the mathematics used in the modelling, and hence specification, of computing systems should be universal in the sense that it be equally suitable for the modelling of natural systems, viz., physical, chemical, biological, or cosmological, and vice-versa, and as a consequence we propose to demonstrate a method of development that promises to lead into a fruitful area of exploration for both research and pedagogy in the general field of dynamically distributed systems.

In the process, we demonstrate that there is a geometry of formal methods, complementary to the usual algebra of formal methods and show how the geometry may be used in practice in the formal specification of systems.

The results are immediately applicable to the model-theoretic formal methods such as VDM and Z.

keywords: algebra, distributed system, formal method, geometry, specification, surgery, VDM, Z, zipper process

1 Prologue

What exactly **is** a distributed system?

“Forget for a while about computers and computer programming, and think instead about objects in the world around us, which act and interact with us and with each other in accordance with some characteristic pattern of behaviour”[10, 23].

The term **distributed system**, used in this paper, is taken to denote a collection of autonomous agents, i.e., objects which behave independently of one another and which communicate with one another. Such a distributed system has an architecture which may be regarded as a dynamically configured network of nodes and edges which represent agents (to be implemented as processes, say) and communication channels, respectively. The dynamism is due to the creation and death of agents, on the one hand, and to the configuration of communication channels, on the other. For the analysis of such a distributed system one may use graph theory to model the network [24, 27].

When we try to think of exemplars of such a distributed system as we have in mind, then we usually mention a single system of mobile telephones/computers, wire-based phones/computers, satellites and other paraphernalia. A second exemplar, which is purely a *Gedanken* experiment, consists of hundreds of thousands (or even millions) of microscopic hovering/mobile computers in a single definitively localized environment, such as a room or park, perhaps, and which are cooperating towards the achievement of a single goal. The third exemplar is, naturally, the World Wide Web of people, programs, and machines, one which we consider to be the paradigmatic exemplar for the next millenium [16]. Finally, there is the exemplar of the system of a particular collection of human beings, not necessarily with contemporaneous coherency but nevertheless thematically bound together in some fashion.

In all cases, it is recognized that in focusing on the key aspects of a distributed system there are only two factors which stand out: behaviour and communication. When we consider the current foundational models of communicating

autonomous agents, such as CCS [19], which is properly called a “process calculus” [19, 4] and CSP [10], etc., it seems clear that communication occurs on a channel that is expressly set up such that information output from one entity a is input by a second (and possibly the same) entity $\bar{a}$. (See, for example, [10, 134]).

Now both CCS and CSP are instances of what are customarily called **process algebras** and it has ever been the goal of the so-called Irish School of the VDM [13], which is founded on classical algebra, to find a natural joining with a process algebra. Furthermore, it is customary to present the mathematical structure of the **monoid** as lying at the heart of computing with certain exceptions reserved for the semigroup.

DEFINITION 1.1 A set A is said to be a semigroup if it is closed under an associative inner law or binary operation, $\star$, i.e., such that for all a, b in A then $a \star b$ is also in A .

A monoid is a semigroup furnished with an identity element for the inner law. In common with others we have always asserted that the monoid does indeed hold central stage in computing and have bemoaned the fact that the group (a monoid with inverse elements), which is one of the most elegant structures in mathematics with profound applications in the natural sciences, seems to have been “outlawed by nature” in the computing sciences.

The **one exception** which we have admitted is the role that the free group plays.

All of computing may be reduced to actions on the free¹ monoid over some alphabet of symbols, Σ , where the inner law is concatenation. It is the foundation of ‘list processing’ languages: LISP, PROLOG, etc., and lies also at the heart of formal languages and their machines (see [6]). We may say that communication has occurred if the symbols a and $\bar{a} = a^{-1}$ annihilate:

$$a\bar{a} = \bar{a}a = 1$$

where 1 denotes the null word, or empty word. Such annihilation of a symbol and its opposite lies at the heart of the free group.

HYPOTHESIS 1.1 The basic mathematical structure which underpins communication is the free group on some alphabet.

It seems reasonable to identify a and $\bar{a}$ with a name and co-name à la CCS [19, 37] and the null word with the silent action τ . This association between CCS semantics and an Abelian free group has already been carried out with respect to a *synchronous* variant of CCS, denoted SCCS and recorded by Milner [19, 195–203] and from which an *asynchronous* calculus can be derived by introduction of an explicit waiting action. It is only recently, that is literally at the time of writing, that we had become aware of this group-theoretic approach through the text of Fencott [7, 274] and, in passing, it is worth noting that the latter’s variants, CCS^{*} and CCS⁺, the semantics of which are given in terms of transitions with respect to a *bag* of actions, may also yield interesting results.

There is no doubt that the algebra of the group-theoretic version of CCS is “somewhat more elegant” [than standard CCS] [19, 197]. That it is not better known may be due to two factors, (a) the unfounded widespread belief that group theory does not truly belong in computing, alluded to above, and (b) the expansion law leads to the classical exponential explosion. Nevertheless, Milner [19, 202] noted that that “with further experience with the synchronous calculus ways may be found to preserve its power without suffering excessive expansion.” Has there been such further experience?

Moreover, ever since learning of the π -calculus [20], with its support for the notions of *locality* and *mobility*, and realizing that the existence of mobile telephony provided a **model** for it, and subsequently having perused the original technical reports [21], [22], it seemed absolutely clear that here was a specification language that would be even better than (S)CCS. However, no evidence of an association between the semantics of the π -calculus and the free-group has yet turned up. Further, in a private conversation (Alexis Donnelly, University of Dublin, Trinity College), we have been shown that it is all too easy to confuse the notation of the π -calculus with the a and $\bar{a}$ of a free group. Therefore, it is pleasing to discover that Hennessy’s recent work on “a fully abstract denotational semantics for the π -calculus” [9, 5] contains a concept of “communication potential”, viz., $n?$ and $n!$, which suggests a possible way forward. In short, using the latter notation for the π -calculus may permit us to build an appropriate independent free group semantics for the communication.

¹The notion of **freeness** is simply a mathematical abstraction for the notion of **basis** of a vector space.

In addressing the problem of the marriage of a model-theoretic method and a process algebra, a problem which has been tackled many times with a varied degree of success by others, viz., the marriage of CCS and VDM-SL [25], we have asked ourselves what sort of conceptual framework *we* needed in order to wed the algebra used in the Irish School of the VDM with that of, say, the SCCS or the π -calculus. Naturally, there was a notation issue; there always is. But more importantly, we wished to have a **process**, an extension of methods already in use, by which we might build models. In particular and to illustrate the sort of questions we ask ourselves, let us address the single issue of the origin or creation of symbols such as a and $\bar{a}$. From whence do they come? In what manner are they generated?

We shall proceed as follows. First we present a hypothesis which compares a distributed system to a chaotic system. It seems to be the right sort of setting or environment for what we have in mind. Second, we shall examine the nature of variables and anti-variables and present the appropriate mathematical framework. This provided the impetus for a device which we call the **zipper**, which proved to be extraordinarily fruitful. Third, we demonstrate how such variables and anti-variables are distributed across a system. Fourth, we introduce the (secondary) notion of time and its (uniquely) forward arrow.

2 Chaos

HYPOTHESIS 2.1 A distributed system is inherently chaotic.

When we say that a distributed system is chaotic, we intend the word ‘chaotic’ to have its current technical meaning in mathematics. Specifically, we shall associate **chaos** with a nonlinear system, that is to say, “a set of nonlinear equations, which may be algebraic, functional, ordinary differential, partial differential, integral or a combination of these” [5, 1]. In the title of the paper we have spoken about a “dynamical distributed system” and a dynamical system is “used as a synonym of nonlinear systems when the nonlinear equations represent evolution of a solution with time or some variable like time” [5, 1].

Given that the intended domain of application is specifically a computing system then it seems reasonable to suppose that we will be using a system of nonlinear algebraic or functional equations. We will obtain the dynamics by using a ‘state-change’ operator which in turn will allow us to derive time itself as a **secondary notion**.

Now we shall attempt to justify the hypothesis.

- A “fixed point is the analogue of an equilibrium point of a system of ordinary differential equations” [5, 68] and we know that minimum fixed points are the solutions of the recursive equations by which we compute.
- The syntactic basis of our programming languages are given by context-free grammars and the recursive equations of a context free-grammar are nonlinear.
- In communication between two processes we are perfectly entitled to suppose that the messages are programs à la Java or TeleScript and such programs can only be interesting if they are words over a nonlinear system.

Although a distributed system is *per se* chaotic, we expect to get useful results by applying certain kinds of controls such as protocols, paradigms of communication, etc.

But how are we to model such chaotic computing systems? At present we suppose that in the message-passing that takes place there are associated words over a free group, some of which are irreducible, i.e., of the form $xy\bar{x}y$, $x\bar{y}xy$, etc. This emphasis on the free group is justified by two further observations: (i) it arises naturally as a generalization of the integers and the process of construction is precisely analogous to the behaviour of a pushdown automaton (to which we will provide a demonstration below) and (ii) it occurs naturally in describing paths in the general abstract spaces of algebraic topology. In fact, it was in perusing a paper of Whitney [26], *Moscow 1935: Topology Moving Toward America*, that the first evidence of a practical geometrical approach to the use of formal methods began to emerge. Specifically, it was the following passage of Whitney which was responsible for the triggering.

I mention just one crucial idea of Hopf: that of “homotopy boundary”, λ , from which everything flowed: From a fixed point P of K , choose a (simplicial) path to any 2-simplex σ^2 , go around its boundary, and back to P :

$$\lambda = \gamma \partial \sigma^2 \gamma^{-1}$$

Now λ (in R) is a relation in the fundamental group G of K , associated with the cell σ^2 , considered as a 2-dimensional chain in K . (Write $G = F/R$, the free group F is the fundamental group of the 1-dimensional part K^1 of K . Now $\lambda \in R \subset F$.)

The key ideas seemed to be that (i) ‘paths are labelled with free group elements’, i.e., words, and (ii) certain ‘non-reducible words have special significance’.

But such lofty notions and putative theories do not suffice. It is also necessary to provide a mechanism, a means whereby the notions translate effectively into algorithms and, hence, into programming language constructs. As mentioned above, to the symbols of the free group, x and $\bar{x}$, already corresponded the channel naming in CCS, and by extension, that of the π -calculus. This was already evident at the VDM’91 Conference in Noordwijkerhout. It was not until the FME’94 Conference in Barcelona that the next step was suggested and to that we now turn.

3 Variables and anti-variables

In their Barcelona paper on **action systems** Back and Sere [2] suggested that “the block notation $\llbracket \text{var } w := w_0 \bullet S \rrbracket$ wraps up a number of different things into a single construct:

- (i) It *associates* the variables w with the statement S ,
- (ii) it *creates* a new variable w on entry to the block,
- (iii) it *initializes* the variable w to w_0 ,
- (iv) it *hides* the variable w from the environment, and finally,
- (v) it *destroys* the variable w on exit from the block.”

They then proposed that in order to “gain more flexibility in parallel composition, and in order to be able to model distributed systems in a realistic fashion,” they needed to deal with the different aspects of the block construct and generalize it in an appropriate fashion.

Instead of following through with their approach, their ideas of creation and destruction of variables prompted a different train of thought. That is to say, the idea of creation and destruction of a variable seemed to agree well with the notion of annihilation at the time of communication. Their paper prompted the sketch (Fig. 1) which was to have major influence on subsequent developments. Specifically, it was hypothesized that the declaration of a variable, say

Figure 1: The Big Bang idea

x , would also immediately give rise to its opposite, $\bar{x}$. Then the destruction of a variable would be accomplished by bringing it together again with its opposite. It was not clear at the time what sort of programming language constructs might be appropriate to capture this notion of the creation and destruction of a variable. What did seem evident was

that the diagram suggested points of contact with the world of physics, specifically of quantum phenomena, and the world of cosmology.

But one ought not to consider oneself restricted to the creation and destruction of variables. It is clear that by extension one might envisage the creation and destruction of objects, processes, etc.

As a first attempt to express the creation (i.e., variable declaration at the leftmost point of the “big bang” in Fig. 1), one might write

$$1 \rightsquigarrow x\bar{x}$$

where, again, the 1 denotes the empty word, i.e., **nothing**! Perhaps a better notation, in order to signify the separation of x and $\bar{x}$, might be

$$1 \rightsquigarrow (x, \bar{x})$$

or even

$$1 \rightsquigarrow \begin{bmatrix} x \\ \bar{x} \end{bmatrix}$$

Supposing that such a pair, $(x, \bar{x})$, would collapse in much the same way that an electron and positron would, we might write

$$(x, \bar{x}) \rightsquigarrow 1$$

The notation is at present suggestive only. The next step was to find a suitable mathematical basis to support the ideas and to broaden them so that they might be used to model real systems.

The “big bang” sketch was originally a nice picture. However, when looked at sideways, it suggested the possibility of a zipper.

3.1 The Zipper Process

The idea of the zipper was simple. When ‘zipped up’ the zipper closed, the opposites came together, and we had annihilation. In addition, the opened zipper suggested a railway siding and railways sidings were once used (and perhaps still are used) to explain stack machines, or pushdown automata. When first used, the zipper was also purely diagrammatic and suggestive (Fig. 2). It was not until this year, 1996, that it was formalized.

Let Σ denote some alphabet of letters $\{a_1, a_2, \dots, a_n\}$. Sequences of letters over Σ are called words and the space of such words is usually denoted by Σ^* . We denote the empty word by 1 and concatenation $(\cdot)$ by juxtaposition. The space of all words under concatenation is usually denoted by $(\Sigma^*, \cdot, 1)$ and is called the **free monoid** (over the alphabet Σ).

For convenience we denote letters of Σ and singleton words by letters at the start of the English alphabet: $a, b, c, \dots$, and other words by letters at the end of the alphabet: u, v, w .

By a **zipper** process, $\zeta: x \rightarrow \bar{x}$, (Fig. 2) we construct the alphabet of formal inverses $\bar{\Sigma} = \{\bar{a}_1, \dots, \bar{a}_n\}$. The idea of using a zipper to denote the desired construction came from (a) the ‘mechanical device’ so-named, and (b) from the big bang picture (Fig. 1) of the creation of a variable. With it we were able to construct, for example, the group of integers, $(\mathbb{Z}, +)$, from the monoid of natural numbers under addition, $(\mathbb{N}, +, 0)$ [16]. We give here a brief sketch of this process.

For each natural number n , we construct its opposite $\zeta(n) = \bar{n}$. Then the operation of addition on $\mathbb{N}$ is extended such that $n + \zeta(n) = 0$. Mathematically, there are some important issues yet to be addressed. First, it is clear that under the zipper process, we obtain a **second zero element**, $\zeta(0) = \bar{0}$. Thus we have a monoid of opposite natural numbers, isomorphic to the original.

To obtain a single zero we must identify 0 and $\zeta(0) = \bar{0}$. This is achieved by **glueing** them together. Such identification (or glueing) is commonplace in topology [1]. For example, one may construct an atlas of the world by glueing

Figure 2: The Zipper process

Figure 3: Glueable maps φ and ψ

together different charts (Fig. 3). In passing we note that the glueing of the zeros above is essentially equivalent to the construction of a fixed-point for the zipper: $\zeta(0) = 0$.

To complete the construction of the integers by the zipper process we must now **reverse the order** on the left hand branch. We do not have space here to elaborate on the especial significance of this reversal of order and of the uses of the zipper process to other spaces and their interpretations.

Let us now turn to the construction of the free group by zippering the free monoid. We have already obtained the formal inverses of letters. By extension, we construct inverse words:

$$\begin{aligned}\zeta(aw) &= \zeta(w)\zeta(a) \\ &= \zeta(w)\bar{a} \\ \zeta(1) &= 1\end{aligned}$$

where we have already glued together the null word 1 and its opposite $\zeta(1) = \bar{1}$. In addition, and in analogy to the construction of the integers, we must also **reverse the order** of letters. Thus, for the lexicographical ordering $a < b$, we have $\bar{a} > \bar{b}$. It is this reversal property of the zipper which tells us that we are dealing with a construction which is ‘isomorphic’ to the pushdown automaton.

Since the Irish School of the VDM is founded on a semantics of monoids and their morphisms [13], [14], then by

construction we have just demonstrated that the zipper function may be added to that base. Specifically, ζ is an anti-isomorphism from the free monoid $(\Sigma^*, \cdot, 1)$ to the free monoid $(\bar{\Sigma}^*, \cdot, 1)$, where we **identify** the null word 1 with its formal inverse $\zeta(1) = \bar{1}$:

$$\begin{aligned}\zeta(uv) &= \zeta(v)\zeta(u) \\ \zeta(1) &= 1\end{aligned}$$

Note also that the zipper function, ζ , is a formal extension of the reverse anti-automorphism, ϱ , on the free monoid:

$$\begin{aligned}\varrho(uv) &= \varrho(v)\varrho(u) \\ \varrho(1) &= 1\end{aligned}$$

If we now extend the definition of concatenation to include annihilation of words of the form $x\bar{x} = \bar{x}x = 1$, then the structure $((\Sigma \cup \bar{\Sigma})^*, \cdot, 1)$ is called the **free group**. Finally, for completeness, we extend the zipper function to the free group itself by defining $\zeta(\bar{a}) = \bar{\zeta(a)} = a$.

Our hypothesis that the free group is the right mathematical basis for the modelling of communication is now considerably strengthened.

The work on the free group and the effectiveness of the zipper process led to some further astonishing results and exhibited some interesting points of contact with algebraic topology. In short a world of geometry of formal methods started to open up. We give one simple example of this development.

3.2 The len homomorphism

At a VDM meeting in Brussels, 20th April 1988, it had been postulated that the tail-recursive forms ψ_w , for an arbitrary word $w \in \Sigma^*$ which correspond to the Σ^* -homomorphisms (see [13])

$$\psi: (\Sigma^*, \cdot, 1) \rightarrow (M, \oplus, u)$$

for an arbitrary monoid $(M, \oplus, u)$ are essentially generalized *affine* transformations. This hypothesis was the first concrete point of contact between the algebra of constructive mathematics and a geometry.

Here we show that the correspondence is exact for the **len** homomorphism. Specifically, we show that there is a quotient group of tail-recursive forms which is isomorphic to a subgroup of the affine group of translations.

Denoting the **len** homomorphism by ψ we note that

$$\begin{aligned}\psi(vw) &= \psi(v) + \psi(w) \\ \psi(1) &= 0\end{aligned}$$

The corresponding constructive case is

$$\psi(ew) = F(e) + \psi(w)$$

where $F: e \mapsto 1$. It may be shown ([13]) that the tail recursive form ψ_w is *connected* to $\psi(w)$ by the initialization

$$\psi(w) = \psi_w(0)$$

and the tail-recursive form is defined by the equations

$$\begin{aligned}\psi_{vw}(m) &= \psi_w\psi_v(m) \\ \psi_1(m) &= m\end{aligned}$$

The constructive case is given by

$$\psi_{ew}(m) = \psi_w(m + F(e))$$

Figure 4: Tail-recursive functions as translations

It is already clear that the tail-recursive forms form a monoid with the inner law given by

$$\psi_w \psi_v = \psi_{vw}$$

and identity element ψ_1 .

When one inquires as to the actual value of the computation of the length of a non-null word w then $\psi(w) = \psi_w(0)$ where $\psi_w(m)$ is given by

$$\psi_w(m) = m + {}^+/F^*w$$

The form $\psi_w \leftrightarrow {}^+/F^*w = n, n \in \mathbb{Z}$, is the composition of an iteration of $F: e \mapsto 1$ followed by a reduction with respect to addition. **It is precisely the form of this result that suggests an affine transformation.**

By zippering we showed how the free group may be obtained from the free monoid. Since both the integers $\mathbb{Z} = (\mathbb{N} \cup \bar{\mathbb{N}}, +)$ and the free group $G = ((\Sigma \cup \bar{\Sigma})^*, \cdot)$ are zippered structures then it is reasonable to expect that for each tail-recursive form ψ_w there is a corresponding (additive) inverse tail-recursive form $\bar{\psi}_w = \psi_{\bar{w}}$. (See Fig. 4). Such a tail recursive form ‘behaves’ exactly like the natural logarithm. Let us now establish the geometrical connection.

An *affine space* may be thought of as “ordinary” space [3, 9]. Specifically, it is very much like a vector space but

Figure 5: Points and displacements in an affine space

without any particular point playing the role of origin. Now a vector space is defined over a field. This is much too strong for our requirements.

Consequently we introduce the weaker, i.e., more abstract or general, structure of a (left) R -module for an arbitrary ring R [12, 163]. Now we are in a position to give a definition of a generalized affine space, basing our definition on that of Crampin and Pirani [3, 9].

DEFINITION 3.1 [Generalized affine space] A set $\mathcal{A}$ is called a generalized affine space modelled on the R -module $\mathcal{M}$ if there is a map called a generalized affine structure, $\mathcal{A} \times \mathcal{M} \rightarrow \mathcal{A}$, denoted additively:

$$(x, m) \mapsto x + m$$

such that

1. $(x + m) + n = x + (m + n)$
2. $x + 0 = x$
3. for any pair of points $x, x' \in \mathcal{A}$ there is a unique element of $\mathcal{M}$, denoted $x' - x$, such that $x + (x' - x) = x'$.

For the problem in hand we choose the affine space to be $\mathbb{Z}$, the set of integers, and since $\mathbb{Z}$ is also a ring, then it will do nicely for our $\mathbb{Z}$ -module.

It is clear that the tail-recursive forms ψ_w satisfy conditions (1) and (2) above. To satisfy (3) we must guarantee uniqueness.

Given two words v and w of the same length, say n , then we have the corresponding tail-recursive forms ψ_v and ψ_w . We construct the obvious natural equivalence relation on these forms, $\psi_v \equiv \psi_w$. To choose a representative for the equivalence class of all forms ψ_w for which the word w has length n , we use the unique sort homomorphism, denoted by σ :

$$\sigma: (\Sigma^*, \cdot, 1) \rightarrow (\Sigma_{\leq}^*, \nabla, 1)$$

where ∇ denotes merge sort [13]. The extension of the sort homomorphism to the corresponding free group is obtained by imposing an ordering on the barred letters suggested directly by the zipper:

$$\dots < \bar{c} < \bar{b} < \bar{a} < a < b < c < \dots$$

and directly in analogy to the ordering on the integers.

Then for the equivalence class of all tail-recursive forms ψ_w such that $\psi(w) = n$, we choose the unique $\psi_{\sigma(w)}$. Now it is easy to demonstrate that we have congruence. Specifically, for elements $\psi_{\sigma(v)}$ and $\psi_{\sigma(w)}$ we must show that the following sum is well-formed, $\psi_{\sigma(v)} + \psi_{\sigma(w)} = \psi_{\sigma(vw)}$. By definition, we have $\psi_{\sigma(v)} = +/F^*\sigma(v)$ and $\psi_{\sigma(w)} = +/F^*\sigma(w)$. Hence

$$\begin{aligned} \psi_{\sigma(v)} + \psi_{\sigma(w)} &= +/F^*\sigma(v) + +/F^*\sigma(w) \\ &= +/F^*(\sigma(v) \nabla \sigma(w)) \\ &= +/F^*\sigma(vw) \\ &= \psi_{\sigma(vw)} \end{aligned}$$

The only interesting aspect of the proof is the observation that $\sigma(vw) = \sigma(v) \nabla \sigma(w)$. Hence, we have constructed a subgroup of the group of tail-recursive forms which is isomorphic to the integers and we are almost done. A simple matter of inclusion $\mathbb{Z} \hookrightarrow \mathbb{R}$ does the rest.

3.3 Interlude

It took a long time between the statement of the original hypothesis that the tail-recursive form of the len homomorphism was essentially a translation of space (April 1988) and the assembly of the mathematics needed to establish and verify it (March 1996). There is now no doubt that this was possible only because we were prepared to review all of our basic assumptions about what constitutes constructive mathematics (not exactly the same as constructivism) and how classical mathematics has more to offer than we hitherto had suspected.

Clearly, the free group is an important keystone. Since a tail-recursive algorithm of the form ψ_w is just a while-loop program on sequences or lists, then we have just demonstrated that not only do ‘negative’ lists exist but also that the corresponding ‘negative’ while-loop programs exist. Of course, to carry out the usual computations, we use the correspondence $\psi_w = \psi_{\bar{w}}$. In some sense we have ‘two parallel universes’, suggested perhaps by the separation in Fig. 1.

When we analyze the method by which we obtained our result, then it is clear that much depends on the concept of the zipper. The zipper was simply a copying function which, in this case, doubled the initial alphabets, the double or copy of a letter being denoted by a barred letter. Such copying or doubling plays a critical role in the Markov algorithm

[17], [18] and we strongly suspect that there is much in the Theory of Markov Algorithms that we might yet use in attempting to grapple with the problem of the construction of an algebra based on the free group for the semantics of, say, communication in the π -calculus.

The idea of negative lists is not new. Just as the integers $\mathbb{Z}$ may be defined in terms of pairs of natural numbers $\mathbb{N} \oplus \mathbb{N}$, so may ‘general’ lists be defined in terms of pairs of lists.

Historically, we began by considering pairs of the form (us, s) , called difference lists and used to construct very efficient programs in PROLOG. Under the transformation

$$(u, v) \mapsto u\bar{v}$$

the difference list (us, s) maps to $us\bar{s} = u$ and the space of all such difference lists forms a monoid [13] under the product

$$(us, s)(vt, t) = (uvts, ts)$$

[**Aside:** In [13] the ‘intuitive’ product $(us, s)(vt, t) = (uvts, ts)$ was given. Although a monoid of positive lists is thus determined, the product does not generalize.] If we use the equivalence relation $(us, s) \equiv (us\bar{s}, 1)$ then we can form the quotient monoid which is obviously isomorphic to the corresponding free monoid. The pair $(u, 1)$ is the canonical representative of ‘positive’ lists.

Similarly, pairs of the form (t, vt) which correspond to $t\bar{v}\bar{v} = t\bar{v} = \bar{v}$, are clearly ‘negative lists’. The canonical representative is $(1, v)$. For the inner law of composition of such negative lists we have

$$(s, us)(t, vt) = (st, vust)$$

There is a natural zipper function on positive lists, $\zeta: (us, s) \mapsto (s, us)$. It is clear that we can use this to demonstrate that the monoids of negative lists are isomorphic to the monoids of positive lists. The end result of all this algebra is simply that we have **potentially** a suitable implementation structure for all algorithms related to the free group.

The final task was to find an inner law of composition for completely arbitrary pairs (u, v) and (s, t) . At first we considered following the analogy of the formation of the integers out of the pair of monoids of natural numbers. All efforts to do so failed abysmally. Now we know why. Indeed, the following law of composition which we discovered in September 1996 and derived directly from the transformation $(u, v) \mapsto u\bar{v}$,

$$(u, v)(s, t) = (u\bar{v}s, t),$$

indicates that it is impossible (due to the occurrence of the negative list $\bar{v}$) to achieve the desired result by ‘joining’ the two monoids of positive and negative lists. In short, the inner law of composition of pairs of lists can only be carried out within the context of the square of a free group, i.e., the cartesian product of a free group with itself! Space prohibits us from elaborating upon this new development here. Instead we terminate the interlude and turn our attention to some of the other developments, however briefly, which add to the conceptual framework (of the dynamically distributed systems) which we are in the process of constructing.

4 Surgery and State Change

We wish to terminate this paper by showing very briefly in what way a fresh look at our assumptions underlying the Irish School of the VDM has contributed to some other very interesting mathematics and, consequently, to some of the new ideas on specification techniques, with particular reference here to the theme of dynamically distributed systems. We begin with the relationship between total and ‘partial’ functions.

The fundamental mathematical structure in computing is the monoid of the space of maps, thought of as ‘partial’ functions, some of which are total, under the override operator:

$$(X \rightarrow Y, \dagger, \theta)$$

For example, if we consider memory, μ , to be a mapping from variables ($x \in X$) to values ($y \in Y$), then the usual meaning of assignment is given by

$$\mu \dagger [x \mapsto y]$$

where the variable x in the memory μ is **overwritten** with the value y . For convenience and mathematical generality, if x is not already present in μ then we agree that $\mu \dagger [x \mapsto y]$ is to be interpreted as the extension of μ by $[x \mapsto y]$.

The override operator must of necessity occur in any formal specification language which deals with **change of state**, viz., VDM, Z, etc.

We may define the override operator in terms of a cut operator, $\Leftarrow$, and a paste operator, $\sqcup$, which are the usual removal and extension of maps, respectively. Thus the expression $\mu \dagger [x \mapsto y]$ may be defined by

$$\Leftarrow_x \mu \sqcup [x \mapsto y]$$

and is interpreted to mean ‘cut or chop out the (old) x information from μ ’ and ‘paste in the (new) x information’. We can be said to be doing surgery on the memory μ and we choose the term **surgery** deliberately in order to tie the concept of override to algebraic topology [1, 162]. We also use the “cut and paste” phrase to indicate that there are applications of surgery/override on Apple Macs that are worth exploring.

4.1 The Totalizer

Given a map μ in the space $X \rightarrow Y$, then the value of μ at $x \in \text{dom } \mu$ is denoted as usual by $\mu(x)$. There is never a question of attempting to evaluate μ at some point not in its domain. For convenience, we denote the subspace of total functions from X to Y by Y^X . Thus for a total function $\mu \in Y^X$, $\mu(x)$ is always well-defined for all $x \in X$.

We frequently use inverse image maps in our specifications. The inverse image of $\mu \in X \rightarrow Y$ is a map $\mu^{-1} \in Y \rightarrow \mathcal{P}'X$, where we are accustomed to use the prime to denote the exclusion of the null set (and more generally to indicate the ‘derivation’ of the semi-group from its monoid). It is customary in mathematics to write $\mu^{-1}(y) = \emptyset$ precisely when y is **not** in the domain of μ^{-1} and this conflicts completely with our definition of map evaluation given above. For years we argued that there really was no problem. But consider the simple map $\mu = [x \mapsto y]$. The inverse is $\mu^{-1} = [y \mapsto \{x\}]$. Considering the latter as a map, then by definition $\mu(\tilde{y})$ is meaningless if $\tilde{y} \neq y$.

The problem is completely resolved if one introduces a constant total function $\emptyset^Y$, which is essentially the constant $\emptyset^Y = \emptyset$, and which we will call the **totalizer** of μ^{-1} (Hughes, Private Communication 1995), (Mac an Airchinnigh, Private Communication 1996). Then consider the expression

$$(\emptyset^Y \dagger \mu^{-1})(\tilde{y}) = \begin{cases} \mu^{-1}(y), & \text{if } \tilde{y} = y \\ \emptyset, & \text{otherwise} \end{cases}$$

We consider this to be a constructive definition even though we have a total function $\emptyset^Y$ on the left (of the override operator) and the whole function is total. The reason is simple—the function on the left is a constant! Such a construction, to our surprise, has already been used by Fencott [7, 64]. Every partial function can be totalized in such a manner where the totalizer is a suitably chosen constant (function). Moreover, conceptually it is equivalent to the notion of the completion of a finite state machine and, thus, has a good pedigree within computing [6, 31]. In addition, the space of all strictly partial functions and the space of their totalized counterparts are isomorphic, a fact which can easily be demonstrated by a **cone construction**, to wit, let z be an element which does not occur in the space Y . Then for each strictly partial function $\mu \in X \rightarrow Y$ we associate the cone $\kappa \in X \rightarrow Y \sqcup \{z\}$, where $\kappa = (z^X \dagger \mu)$. Inclusion of the total functions Y^X shows that $X \rightarrow Y$ is isomorphic to $X \rightarrow Y \sqcup \{z\}$.

4.2 Variable distribution

We have not yet solved completely the problem of the distribution of variables and anti-variables. But we have made considerable progress. A distributed system is essentially a **partition** of some space and inverse images μ^{-1} give rise to partitions of the domains of the maps μ . So important is this relationship between map and partition that it has been

accorded a special name in classical mathematics—that of fibre [12, 12–13]. A very convenient analogy for a bundle of fibres is that of Goldblatt [8]. He likens it to a bunch of asparagus spears. Thus, at the root of each fibre we have the y element. The head or spear is nothing more than $\mu^{-1}(y)$. There is more to the theory of fibre bundles than asparagus spears! A terse overview is given by Schwartz [23, 57–64].

Clearly, we can envisage the construction of a suitable μ such that its fibres give us a distribution of names. Let us consider a special case of this notion of fibre bundle. A hash function h on words $w \in W = \Sigma^*$ is, by definition, a *total distribution function*, a function which constructs partitions (Hughes, Private Communication 1996). Formally, h is an element of the total function space $\mathbb{Z}_p^W$, which is a subspace of $W \rightarrow \mathbb{Z}_p$, where $\mathbb{Z}_p$ is the group of integers modulo prime p . The fibre over some $j \in \mathbb{Z}_p$, denoted $h^{-1}(j)$, is nothing other than the set of words which hash to j . Hughes has developed a considerable amount of algebra based on this single concept of distribution. Although we have not yet worked out all the details of the application of the algebra to the distribution of names we feel that we are close.

But there is more. Already in 1993 we had discovered the existence of a family of monoids which were given the name ‘indexed monoids’ [15]. Essentially, from any monoid $(M, *, e)$, called the base monoid, one could obtain the corresponding indexed monoid $(X \rightarrow M', \otimes, \theta)$, where the new inner law, $\otimes$, was derived from the inner law of the base monoid:

$$\mu \otimes [x \mapsto m] \triangleq \begin{cases} \mu \sqcup [x \mapsto m], & \text{if } \neg \chi_\mu(x) \\ \mu \dagger [x \mapsto \mu(x) * m], & \text{otherwise} \end{cases}$$

The definition was necessitated by the strict requirement that μ be evaluated at a point x in its domain. Thus, for example, did we obtain the monoid of bags as an indexed monoid of the natural numbers under addition. A more complete account of the indexed monoids is [11] and [4], the latter containing some exciting group-theoretical results based on the concept of the **totalizer**.

Now it is clear that if we have some monoidal structure $X \rightarrow \mathcal{P}Y$ which is inherent to an *agent* then we can construct a corresponding indexed monoidal structure $A \rightarrow (X \rightarrow \mathcal{P}Y)'$. This may be interpreted as a family of structures associated with a family of agents and thus, by virtue of the inherency, a necessary facet of a homogeneous distributed system. Further, there is clearly a direct relationship between the fibre bundles and the indexed monoids.

4.3 Time

We turn now to the final aspect of the conceptual framework which we wish to present, that of the problem of time. We have always felt uncomfortable with the fitting of time to our specifications. No matter what we attempted, it always seemed unnatural. Moreover, the attempts of others seemed equally unnatural. In this year, 1996, two fortuitous events occurred. The first we shall explain here. The second was a remark during the Trinity Week Symposium at the University of Dublin, Trinity College, Dublin (May 1996), that in quantum cosmology, **time itself is considered to be a secondary concept, much as pressure and temperature are**. This astonishing view of time accorded well with the mathematics that we began to develop in March 1996, most of which is due to Arthur Hughes.

Our starting point is the **Hughes monoid** of maps

$$(\mathcal{M} \dagger \alpha, \dagger, \alpha)$$

where $\mathcal{M} = X \rightarrow Y$, and $\alpha \in \mathcal{M}$ is an *atom*, i.e., a singleton map, and $\mathcal{M} \dagger \alpha$ denotes the entire space of all maps $\mu \dagger \alpha$ for each $\mu \in \mathcal{M}$. There are $|X||Y|$ atoms in the monoid $\mathcal{M}$.

DEFINITION 4.1 [Hughes Monoid] A Hughes monoid $(\mathcal{M} \dagger \alpha, \dagger, \alpha)$ is constructed from an arbitrary monoid of maps under surgery, $(\mathcal{M}, \dagger, \theta)$, by overriding all elements μ in $\mathcal{M}$ with respect to an atom α in $\mathcal{M}$.

The initially surprising thing about the Hughes monoid is that the identity element is α and not the customary null map. In addition, it is clear that the Hughes monoid is essentially a translation monoid. Let $\alpha = [x \mapsto y]$ be an atom in $\mathcal{M}$. A *coatom* of α in $\mathcal{M}$ is an element of the space of all atoms $\tilde{\alpha} = [x \mapsto \tilde{y}]$ in $\mathcal{M}$ where $\tilde{y} \neq y$. Let $\tilde{\alpha}$ denote a typical coatom of α . Then $\tilde{\alpha} \dagger \alpha = \alpha$. This *absorption* property of an atom with respect to its coatoms may also be

extended to the empty map, $\theta \dagger \alpha = \alpha$. Consequently, an atom ‘eats’ both null map and its coatoms. This prompts us to define the concept of **kernel** of a Hughes monoid with respect to an atom α .

DEFINITION 4.2 [Kernel of a Hughes Monoid] The kernel of a Hughes monoid with respect to an atom $\alpha \in \mathcal{M}$, denoted K_α , is the set of all elements κ in $\mathcal{M}$ which are absorbed by α under override, i.e., such that

$$\kappa \dagger \alpha = \alpha$$

This kernel, K_α , is the *fibre over the identity element* in the target monoid of the override homomorphism. This is the reason for calling it a kernel. It is easy to demonstrate that the Hughes monoid is isomorphic to $(\triangleleft_\alpha \mathcal{M}, \dagger, \theta)$, the monoid reduced by α . Let us define

$$\mathcal{M}_{j+1} = \mathcal{M}_j \dagger \alpha_j$$

where $\mathcal{M}_0 = \mathcal{M}$ and α_j is a suitable atom in $\mathcal{M}_j$. It may be demonstrated that with an appropriate choice of α_j that $\mathcal{M}_j$ approaches a total function in the limit, where no further change can take place. This limit corresponds to a steady state and the entire process seems to agree with the second law of thermodynamics!

Naturally everything depends on the specific choice of the α_j , $j = 1, 2, \dots$. But there is another way to view the entire process. Let us consider some μ and some α , chosen appropriately. Now let us compare μ and $\mu \dagger \alpha$. We are accustomed to think that here we have an ‘update’, a ‘state-change’, both terms carrying strong connotations of time. But suppose that we were to posit the primacy of structures μ and $\mu \dagger \alpha$? Then by definition of the above process there comes a point when no further change can take place. Moreover there are no cycles. We may interpret the process as a basis for the definition of time, one which agrees with the physics, and one which has only one (forward) direction!

Now let us consider a space $\mathcal{M}$ which is subsequently distributed, i.e., indexed or partitioned. Let us choose the atom α and form $\mu \dagger \alpha$ for each μ in the system. The evolution of the system depends on the partition in which α lies.

Were one so bold as to assume that each partition was *analogous* to a representation of a galaxy or a star, then the process $\mu \dagger \alpha$, would suggest that some galaxies or stars would ‘evolve faster’ than others. More importantly the very arbitrariness of the choice of α with respect to the partition would also indicate that one could not know which was evolving faster than another! Surely this is relativity *par excellence*?

5 Epilogue

It is generally now well known that the Irish School of the VDM eschews formal logic and embraces classical algebra as the basis for the specification of computing systems. Consequently, any proofs that must be carried out must conform to classical mathematics. Such a position, whilst in keeping with the view of mathematicians and engineers in general, should not be construed as a rejection of formal logic, or of the endeavours of other Schools of formal methods. Rather, we seek to attain the same goal by a different route.

In this paper we have set forth those ideas which constitute a part of the conceptual framework by which we view the reality of dynamically distributed systems. We do not construe the term ‘distributed system’ in the ultra-narrow sense of a distributed system of processes. Rather do we conceive that much of physical reality is *per se* a collection of classes of distributed systems.

In proposing a new conceptual framework for modelling, we intended to propose a body of mathematics. Said body had to fit the philosophy of constructive mathematics espoused by the Irish School of the VDM. We have already completely determined an engineering-oriented algebraic basis for the modelling of (data) structure with associated operations. We had long sought a compatible and complementary algebra for communications and behaviour. This paper gives a summary of our progress to date.

5.1 Acknowledgements

I gratefully acknowledge the advice and support of the members of the Formal Methods Group in the University of Dublin, Trinity College, who endured much of the material presented in this paper, and in particular I should like to

mention Dr. Claus Pahl who read a very early draft, Alexis Donnelly who cleared up some misconceptions of the π -calculus and Arthur Hughes who provided much of the innovative mathematics. Is it necessary to state that any errors within are entirely those of the author?

A particular note of thanks is due to the anonymous referees who fearlessly highlighted the obvious deficiencies of an earlier draft.

6 Appendix

For convenience, we give here the definitions of those algebraic structures which are necessary for a proper understanding of the len homomorphism as translation. The definition of an affine space is taken from [3, 9]. All other definitions are from [12].

DEFINITION 6.1 [Affine space] A set $\mathcal{A}$ is called an affine space modelled on the real vector space $\mathcal{V}$ if there is a map called an affine structure, $\mathcal{A} \times \mathcal{V} \rightarrow \mathcal{A}$, denoted additively:

$$(x, v) \mapsto x + v$$

such that

1. $(x + v) + w = x + (v + w)$
2. $x + 0 = x$
3. for any pair of points $x, x' \in \mathcal{A}$ there is a unique element of $\mathcal{M}$, denoted $x' - x$, such that $x + (x' - x) = x'$.

DEFINITION 6.2 [Ring] A ring is a structure consisting of a non-empty set R together with two binary operations, $+$ and $\cdot$ in R and two distinguished elements $0, 1 \in R$ such that

1. $(R, +, 0)$ is an Abelian group;
2. $(R, \cdot, 1)$ is a monoid;
3. The distributive laws

$$a(b + c) = ab + ac$$

$$(b + c)a = ba + ca$$

hold for all $a, b, c \in R$.

The set of integers, $\mathbb{Z}$, is a ring.

DEFINITION 6.3 [Left R -module] If R is a ring, a left R -module is an Abelian group M together with a map

$$(a, x) \mapsto ax$$

of $R \times M$ into M satisfying

1. $a(x + y) = ax + ay$
2. $(a + b)x = ax + bx$
3. $(ab)x = a(bx)$
4. $1x = x$

For both the ring R and the Abelian group M we take the set of integers $\mathbb{Z}$. Then the set of integers, $\mathbb{Z}$, is a left $\mathbb{Z}$ -module under this definition.

References

- [1] M. A. Armstrong. *Basic Topology*. McGraw-Hill Book Company (UK), London, 1979.
- [2] R. J. R. Back and K. Sere. From Action Systems to Modular Systems. In Maurice Naftalin, Tim Denvir, and Miquel Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, pages 1–33. Springer-Verlag, Berlin, 1994.
- [3] M. Crampin and F. A. E. Pirani. *Applicable Differential Geometry*. Number 59 in London Mathematical Society Lecture Notes Series. Cambridge University Press, Cambridge, 1986.
- [4] Alexis Donnelly, Dara Gallagher, and Arthur Hughes. On the Inheritance of Monoid Properties in Indexed Structures: A Tale of Three Proofs. Technical Report TCD-CS-96-04, Department of Computer Science, University of Dublin, Trinity College, March 1996.
- [5] P. G. Drazin. *Nonlinear Systems*. Cambridge University Press, Cambridge, 1992.
- [6] Samuel Eilenberg. *Automata, Languages, and Machines, Volume A*. Academic Press, New York, 1974.
- [7] Clive Fencott. *Formal Methods for Concurrency*. International Thomson Computer Press, London, 1996.
- [8] Robert Goldblatt. *Topoi, The Categorical Analysis of Logic*. North-Holland, Amsterdam, revised edition, [1979] 1984.
- [9] Matthew Hennessy. A Fully Abstract Denotational Semantics for the π -Calculus. Technical report, School of Cognitive and Computing Sciences, University of Sussex, UK, June 1996.
- [10] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, London, 1984.
- [11] Arthur P. Hughes and Alexis A. Donnelly. An Algebraic Proof in VDM^* . In Jonathan P. Bowen and Michael G. Hinchey, editors, *ZUM'95, VDM—A Formal Method at Work, Lecture Notes in Computer Science 967*. Springer-Verlag, Berlin, 1995.
- [12] Nathan Jacobson. *Basic Algebra I*. W. H. Freeman, New York, second edition, 1985.
- [13] Mícheál Mac an Airchinnigh. *Ph.D. Thesis: Conceptual Models and Computing*. Department of Computer Science, University of Dublin, Trinity College, Dublin, Ireland, 1990.
- [14] Mícheál Mac an Airchinnigh. Tutorial Lecture Notes on the Irish School of the VDM. In S. Prehn and W. J. Toetenel, editors, *VDM'91, Formal Software Development Methods Volume 2: Tutorials, Lecture Notes in Computer Science 552*, pages 141–237. Springer-Verlag, Berlin, 1991.
- [15] Mícheál Mac an Airchinnigh. Formal Methods & Testing. In *Tutorials of the Sixth International Software Quality Week*, 625 Third Street, San Francisco, CA 94107–1997, May 1993. Software Research Institute.
- [16] Mícheál Mac an Airchinnigh. The Creation of the World in the Third Millenium of our Era. In *Proceedings of Information Technologies and Programming, IT&P'96, Plovdiv*, pages 21–33, Sofia, June 1996. Bulgarian Academy of Sciences.
- [17] A. A. Markov. *Theory of Algorithms*. Israel Program for Scientific Translations, Jerusalem, 1962. Translated by Jacques J. Schorr-Kon and PST Staff from the Russian *Teoriya algorifmov*, Izdatel'stvo Akademii Nauk SSSR, Moskva-Leningrad, 1954.
- [18] A. A. Markov and N. M. Nagorny. *The Theory of Algorithms*. Kluwer Academic Publishers, Dordrecht, 1988. Translated from the Russian by M. Greendlinger.
- [19] Robin Milner. *Communication and Concurrency*. Prentice-Hall International, London, 1989.

- [20] Robin Milner. Concurrent processes as objects. In S. Prehn and W. J. Toetenel, editors, *VDM'91, Formal Software Development Methods Volume 1: Conference Contributions, Lecture Notes in Computer Science* **552**, page 9. Springer-Verlag, Berlin, 1991.
- [21] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I. Technical Report ECS-LFCS-89-85, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, June 1989.
- [22] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part II. Technical Report ECS-LFCS-89-86, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, June 1989.
- [23] Jacob T. Schwartz. *Differential Geometry and Topology*. Gordon and Breach Science Publishers, New York, 1968. Notes by Adil Naoum and Joseph Roitberg.
- [24] Gerard Tel. *Topics in Distributed Algorithms*. Cambridge University Press, Cambridge, 1991.
- [25] W. J. Toetenel. *Ph.D. Thesis: Model Oriented Specification of Communicating Agents*. Department of Computer Science, Technical University of Delft, The Netherlands, 1992.
- [26] Hassler Whitney. Moscow 1935: Topology Moving Toward America. In Peter Duren, Richard A. Askey, and Uta C. Merzbach, editors, *A Century of Mathematics in America*, volume 1 of *History of Mathematics*, pages 97–117. American Mathematical Society, Providence, Rhode Island, 1988.