ISSN 1038–751X

Report 93–7 March 1993

On Functions and Processors: an automata-theoretic approach to concurrency through distributive categories

N. Sabadini and R. F. C. Walters

School of Mathematics and Statistics
The University of Sydney
NSW 2006
Australia

On Functions and Processors: an automata-theoretic approach to concurrency through distributive categories

N. Sabadini and R.F.C. Walters

Dipartimento di Scienza dell'Informazione, Università di Milano, via Comelico, Milan, Italy,

School of Mathematics and Statistics, University of Sydney, N.S.W. 2006, Australia

November 1992

A calculus of functions, processors, and corresponding processes, is developed in automata-theoretic terms, using ideas described in Walters [W2], [CCS] and Khalil-Walters [KW]. It is used to analyze some notions on resource and refinements expressed in [Sa], [KMS].
The main idea is that to have a structured theory of the construction of automata with data types it is necessary to consider two sorts of things - automata and functions (between data types), and operations involving these two sorts.
The operations on functions yielding functions are those arising from the categorical properties of the cartesian product and the disjoint union of sets, and the distributive law between them: that is, the operations are those of a *distributive category*.
As usual, by an automaton or processor we mean an alphabet Σ, a (not necessarily finite) set X, the *state space*, and, to each letter $a \in \Sigma$, a function $\alpha_a : X \to X$ called the *action* of a on X. Since the actions of processors are functions the operations on functions can be used to construct automata out of functions or other processors.
Finally, a processor whose alphabet consists of a single letter, and whose state space is of the form $X + U + Y$ may yield by iteration a function $X \to Y$; such processors we call *functional*, following Khalil-Walters [KW]. A precursor of this notion occurs in Elgot's iterative theories [E].
Although it is straightforward in this calculus to deal with many issues such as communication between processors, birth and death of processes, parametrized families of processes, and distributed algorithms, we concentrate in this paper on giving first a simple outline of the calculus and then investigating a notion of *refinement* of processor based on the idea of *interpretation* of one processor in another which allows us to discuss resource issues.
In explaining our calculus we make comments on its relation with other automata-theoretic models of concurrency: Petri nets [P], asynchronous automata [Z] and port automata [SAM].

1. Introduction

This paper introduces a new automata-theoretic approach to the problems of concurrency and distributed processing.

It follows the classical style of automata theory and formal language theory where the main issue is the relation between automata and their behaviour. More precisely, we are concerned with methods of construction of new automata out of old, and how the behaviour of the new automata relates to the behaviours of the old. We observe that there is no simple general correspondence between the operations on processors and operations on processes which might allow one to ignore processors in favour of processes. Other theories such as the process algebras [H], [M], and trace theory [Maz] tend to place observations/processes ahead of processes. Of course, on the firm foundation of state and behaviour, abstractions are necessary to analyse state and behaviour.

What is particularly novel in our approach is the ***method of construction*** of automata considered; we have a two-sorted theory, with operations between the sorts, the two sorts being: (1) deterministic automata on finite alphabets, and (2) functions.

The operations for constructing from functions both new functions and the actions of letters on state, are the operations available in a distributive category; that is, a category with sums and products and the law of distribution of products over sums The operations on alphabets considered are also product and sum. The remaining method of construction is that a processor (with a single-letter alphabet and) with state space of the form $X + U + Y$ may compute by iteration a function from X to Y. We call such processors ***functional***.

A precursor to the notion of functional processor is contained in Elgot [E] (and is the basis of the notion of *iterative theory*). Elgot considers functions of the form $\phi : Z \times n \cdot I \to Z \times (m+n) \cdot I$; in other words functions of the form $\phi : X \to X + Y$ (for very particular X, Y). Such a function can be extended to a loop $\psi : X + Y \to X + Y$. Elgot considers the *partial function* from X to Y obtained by iterating ψ. This is a crucially weaker notion than ours because (i) the particular forms of the sets X and Y don't allow the general use of sums and products, and hence data types, to be used, and (ii) since there is no explicit mention of the local state U (and to find it is undecideable) Elgot is unable to hide state in his calculus – a fundamental feature of our model. Finally in Elgot's calculus he is unable to consider the operation of the product of two pseudofunctions, the most interesting operation from our point of view.

Automata constructed in the above way from basic data types seem to be the appropriate ones for the consideration of concurrency and distributed processing; the necessity for iteration is clear; the operations of sum and product are essential for any such consideration since they express, respectively, total conflict and parallelism. The problem of a theory for combining asynchronous machines has preoccupied many authors in recent years (see for example [Wi2], [Pi]). We discuss in a very straightforward way different products of machines – interleaved, synchronous, and asynchronous. The differences between these corresponds to operations on the alphabets. Zielonka's asynchronous automata [Z] fit in our setting in a very clear way; in fact we have a more explicit description of the way in which they are asynchronous machines. We also describe briefly a new type of asnchronous machine obtained by generalizing flowcharts to have distributed control; the machines are reminiscent of Petri nets.

In explaining our calculus we describe some relations with other automata-theoretic

models of concurrency: Petri nets [P], asynchronous automata [Z] and port automata [SAM]. In the full paper we will also discuss the automata of [LS].

The mathematical background needed for this paper is a knowledge of category theory up to products and sums, and the distributive law between products and sums, such as contained, for example, in the first three chapters of Walters [CCS]. Other useful references for the categorical constructions on functions used in this paper are [BR], [Mac]. There are relations at a categorical level between this work and [AM].

2. Operations on functions

The aim of this section is to introduce the operations we will use for producing new functions out of old. For details and the notation used see Walters [CCS].

The category of sets and functions is a *distributive* category; [CCS]; that is it has finite products and sums (=coproducts=disjoints sums) and the canonical function $X \times Y + X \times Z \to X \times (Y + Z)$ is an isomorphism. In such a category there are the following special arrows: the projection maps $proj_1 : X \times Y \to X$, $proj_2 : X \times Y \to Y$; the injections $inj_1 : X \to X + Y$, $inj_2 : Y \to X + Y$; the *diagonal* and the *codiagonal* functions $\Delta : X \to X \times X$, $\nabla : X + X \to X$; Further, given two functions $f : X_1 \to Y_1$, $g : X_2 \to Y_2$, there are the functions $f+g : X_1+X_2 \to Y_1+Y_2$, $f \times g : X_1 \times X_2 \to Y_1 \times Y_2$. Of course the composition of arrows are arrows.

Remark 2.1. What we have described above is a precise family of operations for producing new functions out of old. In more detail, if we have a family of sets $X, Y, Z, \ldots$, (call them the *basic* sets, then consider all the sets which can be obtained from them, and $\emptyset$, I, by repeated use of sum and product (call them the *derived* sets). Now suppose we have a family G of (*basic*) functions between derived sets. Any of the operations above produces from these functions a new function between derived sets. The functions so obtained by repeated use of these operations, beginning with G, we call the *derived* functions.

In everything that follows there will be a given, usually unspecified, collection of basic sets and all functions will be between sets derived from these given ones.

Notation 2.1. If $f : X \to Y$ and $g : X \to Z$ and $h : Z \to Y$ are functions then we denote the composite $\nabla \circ (f + h) : X + Z \to Y + Y \to Y$ by $(f|h) : X + Z \to Y$; and we denote the composite $(f \times g) \circ \Delta : X \to X \times X \to Y \times Z$ by $(f, g) : X \to Y \times Z$. Some other arrows it is useful to have names for: $twist : X \times Y \to Y \times X$ for $(proj_2, proj_1)$ and $twist : X + Y \to Y + X$ for $(inj_2|inj_1)$.

3. Processors from functions

Now let's make precise the notion of processor we intend to use in this paper.

Definition 3.1. A *processor*, P, with alphabet Σ, relative to a finite set of given basic sets, is a set X, the *state space of* P, built up using sums and products from the sets in the collection, and a function $\alpha : \Sigma \times X \to X$. Then if $a \in \Sigma$, the function $\alpha_a : X \to X$ which takes $x \in X$ to $\alpha(a, x)$ is called the *action* of the letter a on the state space.

We shall sometimes denote the process P by the pair (X, α). A *process* of the processor is an infinite sequence of letters $a_0, a_1, a_2, \ldots$ and states,

$$x_0 \mapsto x_1 \mapsto x_2 \ldots \mapsto x_n \mapsto \ldots$$

such that, for each n, $x_{n+1} = \alpha_{a_n}(x_n)$.

At first glance it may appear that we are speaking about interleaving here, without the possibility of simultaneous action, but as we will see in section 4 this is by no means the case since also the action alphabet may have a structure. We believe in a true concurrency approach, since for us interleaving arises from questions of implementation (representing one machine in another).

An important special case is when the alphabet Σ consists of one element. In this case the process consists of a state space X and a single loop $\alpha : X \to X$.

In the following if no alphabet is mentioned in referring to a process it is intended that the alphabet has one letter which we do not name.

It is clear, as we remarked above, that since the actions of processors are themselves special functions between derived sets they can be constructed using operations on functions. We give an examples of constructing a processor (with single letter alphabet) from the basic set $\mathbb{N}$ and the two functions $pred : \mathbb{N} \to I + \mathbb{N}$ and $succ : I + \mathbb{N} \to \mathbb{N}$ using the operations on functions given in the last section.

Example 3.1. We can construct an interesting processor with state space $\mathbb{N}^2 + \mathbb{N}$ as follows. First let $f : \mathbb{N} \to \mathbb{N}^2 + \mathbb{N}$ be the composite of

$$\mathbb{N}^2 \xrightarrow{pred \times 1_N} (I + \mathbb{N})\mathbb{N} \xrightarrow{\cong} \mathbb{N} + \mathbb{N}^2 \xrightarrow{twist} \mathbb{N}^2 + \mathbb{N}$$

$$\mathbb{N}^2 + \mathbb{N} \xrightarrow{1_N \times (succ \circ inj_2) + 1_N} \mathbb{N}^2 + \mathbb{N}$$

It is easy to see that $f(m, n) = (m - 1, n + 1)$ if $m > 0$, and $f(0, n) = n$. The processor we define is then $(f|1_N) : \mathbb{N}^2 + \mathbb{N} \to \mathbb{N}^2 + \mathbb{N}$. One particular process of this processor, with initial state in the first summand of the state space, is

$$(5,3) \mapsto (4,4) \mapsto (3,5) \mapsto (2,6) \mapsto (1,7) \mapsto (0,8) \mapsto 8 \to 8 \mapsto \cdots.$$

In fact, this processor is of the type we will call functional because if the initial state is (m, n) in the first summand, then eventually the process stabilizes at the value $m + n$ in the third summand, and so defines the function $add : \mathbb{N}^2 \to \mathbb{N}$.

4. Simple operations on processors

We give immediately some very basic examples of operations on processors.

4.1. *Parallel processors*

Given two processors P, Q, with alphabets Σ_1, Σ_2, state spaces X, Y, actions α, β, respectively, we can form a new processor $P \times Q$, and alphabet $\Sigma_1 + \Sigma_2$ as follows:

state space $X \times Y$, and action $\gamma_a = \alpha_a \times 1_Y : X \times Y \to X \times Y$ if $a \in \Sigma_1$, and $\gamma_b = 1_X \times \beta_b : X \times Y \to X \times Y$ if $b \in \Sigma_2$.

It is clear that the processes of this new processor consist of ***interleaved*** processes of the two processors.

$\Sigma_1 + \Sigma_2$ is not the only alphabet which acts sensibly on $X \times Y$. Another choice is $\Sigma_1 \times \Sigma_2$ with action $\delta_{(a,b)} = \alpha \times \beta$. Processors of this processor are synchronized parellel processes of P and Q.

Another choice is $\Sigma_1 \times \Sigma_2 + (\Sigma_1 + \Sigma_2)$ with the actions obtained from the last two examples. The new processor might be thought of as an asynchronous parallel product. Trace theory [Maz] is concerned with the fact that the action of $(\Sigma_1 \times \Sigma_2 + (\Sigma_1 + \Sigma_2))^*$ on $X \times Y$ factors through a free partially commutative monoid on $\Sigma_1 + \Sigma_2$.

The asynchronous automata of Zielonka [Z] fit precisely into this picture. Such automata have a global state space which is a finite product $X_1 \times X_2 \cdot X_n$, and to each subset U of $\{1, 2, \ldots, n\}$ an alphabet Σ_U which affects exactly the factors indexed by the subset, and are the identity on the other factors. What really acts on the automaton is the alphabet which is the sum of the sets of the form Σ_U, and $\Sigma_U \times \Sigma_V$, (U, V subsets of $\{1, 2, \ldots, n\}$, U disjoint from V).

With similar ideas, beginning from flow charts we can form a class of asynchronous automata with distributed control (that is with both sums and products in the state space) reminiscent of Petri nets, and with simple methods of combination. The precise description will be given in the full paper.

4.2. *Totally conflicting processors*

Again we could consider processors with alphabets as in the last example but for simplicity we restrict this example to one-letter alphabets.

Given two processors P_1, P_2, we can form a new processor $P_1 + P_2$ as follows: state space $X_1 + X_2$, and action $\alpha_1 + \alpha_2 : X_1 + X_2 \to X_1 + X_2$. It is clear that the behaviours of this new processor is either a process of the first or a process of the second processor.

Notice that totally conflicting action commutes in the same way as parallel action so the commuting is not a defining characteristic of parallel action. (Zielonka's theorem [Z] shows how commuting action can be made parallel by enlarging the system.)

4.3. *Sequential composition of processors*

Given two processors P, Q, with state spaces $X + U + Y$ and $Y + V + Z$, and actions α, β, respectively, with the property that $\alpha(y) = y$, ($y \in Y$), and $\beta(z) = z$, ($z \in Z$), we can form a new processor $P;Q$ as follows: state space

$$X + (U + Y + V) + Z,$$

and action

$$\gamma = (\alpha + 1_{V+Z})(1_{X+U} + \beta).$$

The processes of this new processor $P;Q$ with initial state in X are exactly either those processes of P with no element in Y, or those sequences

$$x_0 \mapsto x_1 \mapsto x_2 \ldots \mapsto x_n = y_0 \mapsto y_1 \mapsto y_2 \ldots$$

where $x_0 \mapsto x_1 \mapsto x_2 \ldots \mapsto x_n$ is the initial segment of a behaviour of P with x_n being the first element in Y, and $y_0 \mapsto y_1 \mapsto y_2 \ldots$ is a process of Q.

Remark 4.1. Let's consider very briefly how one might model a process with an input port, and a queue of messages. Firstly the state space of the process itself must be of the form $X = (N + R \times M)$ where the summand $R \times M$ corresponds to the desire to receive a message of type M; suppose it has action α. Let Q be the set of queues of type M. Then the process together with the queue has state space $X \times Q \cong NQ + RMQ$. The action of the combined system may be split up into the cases corresponding to the different summands: on NQ it is $\alpha \times 1_Q$; on RMQ it is a function obtained by first dequeuing $(Q \to I + QM)$, replacing the contents of M by the message if there is one, and then applying α. A process with an output port has a similar model, and similarly two processes with a communicating channel. The port automata of [SAM] are automata with a family of input and output queues, and can be described in our calculus.

5. Functions from processors — pseudofunctions

Some processors (with single letter alphabets) are intended to compute functions by iteration.

This notion was made precise in Khalil-Walters [KW], where the following definition of pseudofunction was introduced.

Definition 5.1. A *functional processor* or *pseudofunction* from X to Y is a processor with state space of the form $X + U + Y$, and action α such that

(i) for y in Y, $\alpha(y) = y$;
(ii) for each x in X there exists a natural number n_x such that $\alpha^{n_x}(x) \in Y$.

A pseudofunction $\alpha : X + U + Y \to X + U + Y$ defines a function $\mathbf{call}(\alpha) : X \to Y$ (by iteration) as follows:

$$\mathbf{call}(\alpha)(x) = \alpha^{n_x}(x).$$

We often denote the pseudofunction α as $\alpha : X \dashrightarrow Y$ to indicate that it computes a function from X to Y. We call U the set of *local* states of the pseudofunction α, or the *hidden* states of the function $\mathbf{call}(\alpha)$.

Remark 5.1. A pseudofunction α is just a process which for initial states in a certain explicitly part of the state space (X in the above) reaches a stable value in another explicitly given part of the state space (Y in the above), and indicates that it has reached a stable value. Such a process can then be used as a function in building up further functions and processes. Further part of the state space of α, U in the above, is hidden, and the steps of the iteration are hidden when we consider the function $\mathbf{call}(\alpha) : X \to Y$ rather than α, so that $\mathbf{call}(\alpha)$ becomes *atomic*. As you would expect, and we will see this explicitly later, the operation of hiding and making atomic has conceptual advantages and efficiency disadvantages.

Remark 5.2. The next proposition, which we give without proof, contains the basic algebra of **call** — how it relates to the operations on functions and to itself. The proposition justifies the following remarks. Given a set G of basic functions consider the functions obtainable from then by the operations on functions given in section 2, ***together with*** the operation **call**. These the functions are said to be ***computable*** from G (in [SVW2] we show that the functions computable from ***succ*** and ***pred*** are exactly the total general recursive functions). Then if we add to G any of the computable functions (hence considering them as atomic) to form a new set G' of basic functions we do not get any new computable functions. That is, the functions computable from G' are the same as those computable from G. The proposition also implies that a computable function may always be obtained by operations on functions plus a single use of **call**. Another consequence is that computable functions form a distributive category.

Proposition 5.1. The following properties hold for **call**

(i) For any function $f : X \to Y$ then

$$f = \mathbf{call}[(f|1) : X + Y \to X + Y];$$

(ii) If $\alpha : X \to\!\!\!\!\!+\; Y$ and $\beta : Y \to\!\!\!\!\!+\; Z$ are pseudofunctions, with local states in U and V respectively, then

$$\mathbf{call}(\beta) \circ \mathbf{call}(\alpha) = \mathbf{call}[(1_{X+U} + \beta) \circ (\alpha + 1_{V+Z}) : X + W + Z \to X + W + Z]$$

where $W = U + Y + V$.

(iii) If $\phi : X \to\!\!\!\!\!+\; Y$ and $\psi : X' \to\!\!\!\!\!+\; Y'$ are pseudofunctions with local states U and U' respectively, then

$$\mathbf{call}(\phi) + \mathbf{call}(\psi) = \mathbf{call}(a^{-1} \circ (\phi + \psi) \circ a)$$

where a is an isomorphism

$$(X + X') + (U + U') + (Y + Y') \xrightarrow{\cong} (X + U + Y) + (X' + U' + Y');$$

(iv) If $\phi : X \to\!\!\!\!\!+\; Y$ and $\psi : X' \to\!\!\!\!\!+\; Y'$ are pseudofunctions with local states U and U' respectively, then

$$\mathbf{call}(\phi) \times \mathbf{call}(\psi) = \mathbf{call}(b^{-1} \circ (\phi \times \psi) \circ b);$$

where b is an isomorphism(given by the distributive law)

$$XX' + (XU' + XY' + UU' + UY') + YY' \xrightarrow{\cong} (X + U + Y)(X' + U' + Y');$$

(v) (wheels within wheels) Suppose that $\phi : X + U + X \to\!\!\!\!\!+\; X + U + X$ is a pseudofunction with local state W, and $\mathbf{call}(\phi) : X \to\!\!\!\!\!+\; X$ is a pseudofunction. Consider $\psi : Z = X + (U + X + W + X + U) + X \to X + (U + X + W + X + U) + X = Z$ defined by

$$\psi =$$

$$Z \xrightarrow{\phi} (X + U + X) + W + (X + U + X) \xrightarrow{1_{X+U+X+W} + inj + 1_X} Z$$

where inj is the injection of $X + U$ as the first component of Z then

$$\mathbf{call}(\mathbf{call}(\phi)) = \mathbf{call}(\psi).$$

6. Refinement

6.1. *Refinement of processors*

What does it mean for two processors $P = (X, \alpha)$ and $Q = (Y, \beta)$ to go at ***different speeds***? The abstract notion of processor/automaton has no notion of speed; just state and transition. To be able to compare speeds of processors and hence match them in order to get efficient parallel processors, we need the notion of ***refinement***.

Definition 6.1. A *refinement* of processor $P = (X, \alpha)$ is another processor R with state space of the form $Z = X + U$ and action γ, say, such that given a process $z_0 \mapsto z_1 \mapsto \cdots$ of R, with $z_0 \in X$, the subsequence consisting of those states in X is a process of P. We call the summand U the *local* state of the refinement R.

Given the fact that the processes of P are determined by their initial states it follows that every process of P arises in this way from one of R. It is clear now that it makes sense to compare the speeds of P and R; clearly R runs faster (or the same) as P.

It is immediately clear that if $\phi : X \dashrightarrow X$ is a pseudofunction with local state U then the process $(X, \mathbf{call}(\phi))$ has a refinement with local state $X + U$: just take the action of the refinement to be the composite of

$$X + X + U \xrightarrow{\nabla + 1_U} X + U \xrightarrow{inj} X + U + X \xrightarrow{\phi} X + U + X$$

$$X + U + X \xrightarrow{(1_{X+U} | inj_1)} X + U \xrightarrow{inj} X + X + U.$$

Now suppose we have two processors P, Q with refinements P_1 with state space $X_1 = X + U$, Q_1 with state space $Y_1 = Y + V$, respectively (and actions α_1, β_1 respectively). We can now compare the parallel product $P \times Q$, which takes the actions of P and Q as atomic, with the parallel product $P_1 \times Q_1$, in which the actions of P and Q are now not atomic: P is allowed to proceed at a different speed than Q.

The product of the refined processors has state space

$$X_1 \times Y_1 \cong XY + XV + UY + UV$$

and action $\alpha_1 \times \beta_1 : X_1 \times Y_1 \to X_1 \times Y_1$. That part of the state space in which we get a state of P *or* Q is $XY + XV + UY$. If for a particular initial value the processor P_1 spends less time in its local state than Q_1 then the states of the processor P will occur more frequently than those of Q in $P_1 \times Q_1$, usually in the summand $X \times V$.

Notice that having refinements allows one to match parallel processors better. In the parallel product $P \times Q$ each single action of P occurs synchronized with each single action of Q, even if we know by a refined analysis that P proceeds faster than Q.

Notice that our notion of refinement differs markedly from notions currently being considered in Petri nets and process algebra [DG]. In particular it is not at all necessary that a refinement of two parallel processors be parallel, or a refinement of conflicting

processors be conflicting in all steps. In fact in our opinion the idea of refinement should be to introduce information; that is to change our description of the system. A correct refinement should only maintain the same relations on the states of the non-refined process. Processes of the refined system when *restricted* to the unrefined machine should be processes of the unrefined machine. A detailed discussion will be given in the full paper.

We remark that refinement being a notion of interpretation of one machine in another allows us to consider resources; for example the fact that a refinement of two parallel processors may not require the full resources of the product.

Remark 6.1. It is clear that refinements form a category. In fact refinements are a special case of the following notion of morphism between processors. A morphism from $P = (X, \alpha)$ to $Q = (Y, \beta)$ is a function $\mu : X \to Y$ satisfying the property that for each $x \in X$ there is an n_x such that $\beta^{n_x}(\mu(x) = \mu(\alpha(x))$. (Each step of P corresponds to some number of steps of Q.) Notice that in this language there is a morphism from the function computed by a pseudofunction (considered as a one step action) to the pseudofunction.

7. Acknowledgements

We would like to thank Felice Cardone, Mike Johnson, Stefano Kasangian, Wafaa Khalil, Stephen Ma, Giancarlo Mauri, Marco Rodolfi, and Sebastiano Vigna for useful discussions.

In particular, the second author would like to thank Eric Wagner for conversations; Eric has made some similar investigations into the theory of processes using distributive categories. The first author would like to thank Martina for accepting with patience her mamma's work.

This work has been supported by the Australian Research Council, Esprit BRA ASMICS, Italian MURST 40

REFERENCES

[AM] Arbib M., Manes E., *Algebraic Approaches to Program Semantics*, Springer-Verlag, 1986.

[BR] Burstall D. E., Rydeheard R. M., *Computational Category Theory*, Prentice-Hall, 1988.

[DGR] Degano P., Gorrieri R., Rosolini G., A categorical view of process refinement, *Quaderno* 77 Universita degli studi di Parma (1992), 1-16

[E] Elgot C.C., Monadic computation and iterative algebraic theories *Studies in Logic and the Foundations of Mathematics* **80** (1975), 175-230.

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

[KMS] S.Kasangian, G.Mauri, N. Sabadini, *Traces and pomsets: a categorical view*, Tech. Rep. BRA Esprit DEMON , N.193, 1991, versione finale in preparazione

[KW] Khalil Wafaa, Walters R. F. C., An imperative language based on distributive categories II, *to appear* RAIRO Informatique Theorique et Applications

[LS] Lynch N. A., Stark E. W., A proof of the Kahn Principle for input/output automata, *Information and computation* **82** (1989), 81-92

[Mac] Mac Lane S., *Categories for the Working Mathematician*, Springer-Verlag, 1971.

[Maz] Mazurkiewicz A. Concurrent program schemes and their interpretations *DAIMA Rep. PB-78* Aahrus University (1977).

[Mi] Milner A. J. R. G. *Communication and concurrency* Prentice Hall, 1989.

[P] Petri C. A., Concepts of net theory, *Mathematical foundations of concepts of computer science* High Tatras (1973), 137-146

[Pi] Pighizzini G., On the iteration of trace languages, preprint

[R] Reisig W. *Petri Nets: an introduction* EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1985.

[Sa] Sabadini N., Remarks on concurrency and resources

[SW] Sabadini N., Walters R.F. C., On resources, *in preparation.*

[SVW1] Sabadini N., Vigna S., Walters R.F.C., Finitely-generated automata, *in preparation.*

[SVW2] Sabadini N., Vigna S., Walters R.F.C., A note on recursive functions, *in preparation.*

[SAM] Streenstrup M., Arbib M. A., Manes E. G., Port automata and the algebra of concurrent processes

[W1] Walters R. F. C., Datatypes in distributive categories, *Bull. Austral. Math. Soc.* **40** (1989), 79–82.

[W2] Walters R. F. C., An imperative language based on distributive categories, *Mathematical Structures in Computer Science*, **11**, (1992), 1-8

[CCS] Walters R.F.C., *Categories and Computer Science*, Carslaw Publications (1991), Cambridge University Press, (1992).

[Wi] Winskel G.,, Event structures, *Petri nets: applications and relationships to other models of concurrency* Springer Lecture Notes in Computer Science **255** (1987), 325-392

[Wi2] Winskel G.,, Petri nets, algebras, morphisms, and compositionality, *Information and Computation* **72** (1987), 197-239

[Z] Zielonka W. Notes on finite asynchronous automata *RAIRO Inf. Theor.* **27** (1985), 99-135

THE UNIVERSITY OF SYDNEY
Research Reports of the School of Mathematics and Statistics†

92–1	G. M. Kelly	*On clubs and data-type constructors*	January 1992
92–2	Jie Du and Leonard Scott	*Lusztig conjectures, old and new, I*	January 1992
92–3	Richard Dipper and Jie Du	*Harish-Chandra vertices*	January 1992
92–4	A. Dimca and L. Paunescu	*Real singularities and dihedral representations*	February 1992
92–5	K. F. Lai	*On the spectral expansion of a relative trace formula*	February 1992
92–6	Tzee-Char Kuo	*Singularities of real analytic functions*	February 1992
92–7	Tzee-Char Kuo	*Stratification Theory*	February 1992
92–8	Tzee-Char Kuo	*Truncation of Taylor Series*	February 1992
92–9	Aurelio Carboni, Stephen Lack and R. F. C. Walters	*Introduction to Extensive and Distributive Categories*	March 1992
92–10	G. I. Lehrer	*Rational tori, invariant functions on a Lie algebra and the nilpotent variety*	March 1992
92–11	Shu-Hao Sun	*New characterizations of biregular rings and their applications to duality*	March 1992
92–12	J. A. Hillman and C. Kearton	*Algebraic invariants of simple 4-knots*	March 1992
92–13	Jie Du	*IC Bases and Quantum Linear Groups*	April 1992
92–14	Shu-Hao Sun	*Duality on compact prime ringed spaces*	April 1992
92–15	R. B. Howlett and G. I. Lehrer	*On Harish-Chandra induction and restriction for modules of Levi subgroups*	April 1992
92–16	Jonathan A. Hillman	*On 4-manifolds with universal covering space $S^2 \times S^2$, $S^2 \tilde{\times} S^2$, S^4 or CP^2*	May 1992
92–17	Roman R. Poznanski	*Electrotonic length of neurons with complex branching dendrites*	June 1992
92–18	Mary C. Phipps	*A Note on the Cramér-Rao Lower Bound*	June 1992
92–19	Mary C. Phipps	*The Error Performance of a Communication System Using List Detection*	June 1992
92–20	A. Dimca, J. A. Hillman and L. Paunescu	*On hypersurfaces in real projective space*	June 1992
92–21	Brigitte Brink and Robert Howlett	*A finiteness property of Coxeter groups*	July 1992
92–22	Susan B. Niefield and Shu-Hao Sun	*Algebraic De Morgan's Laws for Noncommutative Rings*	July 1992
92–23	J. A. Hillman	*On 4-manifolds homotopy equivalent to surface bundles over the projective plane*	July 1992
92–24	Adam Parusiński	*Lipschitz stratification of subanalytic sets*	July 1992
92–25	Adam Parusiński	*Lipschitz Stratification — A Review of Results*	July 1992
92–26	Laurentiu Paunescu	*V-Sufficiency from the weighted point of view*	July 1992
92–27	Donald I. Cartwright, Anna Maria Mantero, Tim Steger and Anna Zappa	*Groups acting simply transitively on the vertices of a building of type $\tilde{A}_2$ I.*	July 1992

†Editor: Dr D. E. Taylor

.../

92–28	Donald I. Cartwright, Anna Maria Mantero, Tim Steger and Anna Zappa	*Groups acting simply transitively on the vertices of a building of type $\widetilde{A}_2$ II: the cases $q = 2$ and $q = 3$*	July 1992
92–29	G. M. Kelly and Stephen Lack	*Finite-product-preserving functors, Kan extensions, and strongly-finitary 2-monads*	August 1992
92–30	G. M. Kelly, Stephen Lack, and R. F. C. Walters	*Coinverters and categories of fractions for categories with structure*	August 1992
92–31	Yi Cao	*On the non-negative integer solution of the equation $N = ax + by + cz$*	August 1992
92–32	D. Easdown and W. D. Munn	*On semigroups with involution*	August 1992
92–33	R. R. Poznanski	*Nonlinear Summation of Junction Potentials in a Three-Dimensional Bisyncytium*	August 1992
92–34	Jonathan A. Hillman	*On 4-manifolds with finitely dominated covering spaces*	August 1992
92–35	Jie Du	*Canonical bases for irreducible representations of quantum GL_n, II*	September 1992
92–36	Shu-Hao Sun	*One more sheaf representation theorem for modules*	September 1992
92–37	Jonathan A. Hillman	*Minimal 4-manifolds for groups of cohomological dimension 2*	September 1992
92–38	Adam Parusiński	*Limits of Tangent Spaces to Fibres and the w_f Condition*	September 1992
92–39	Duong Phan	*Some explicit twistfree vacuum spacetimes with a spacelike Killing vector*	September 1992
92–40	G. I. Lehrer et J. Thévenaz	*Sur la conjecture d'Alperin pour les groupes réductifs finis*	October 1992
92–41	D. I. Cartwright, W. Mlotkowski and T. Steger	*Property (T) and $\widetilde{A}_2$ groups*	October 1992
92–42	Laurentiu Paunescu	*The homotopy type of the real Milnor fiber*	October 1992
92–43	Koo-Guan Choo and Shu-Hao Sun	*On extensive normal quantales*	October 1992
92–44	Wafaa Khalil	*Functional Processors*	October 1992
92–45	Laurentiu Paunescu	*The topology of the real part of a holomorphic function*	December 1992
92–46	Wafaa Khalil	*Composing Functional Processors*	December 1992
93–1	Amitavo Islam and Wesley Phoa	*A category-theoretic model for relational databases with heterogeneous data*	January 1993
93–2	S. P. Glasby	*Parametrized polycyclic presentations of minimal soluble groups*	January 1993
93–3	S. P. Glasby	*On the minimal soluble groups of derived length at most six*	January 1993
93–4	J. A. Hillman	*The Algebraic Characterization of Geometric 4-Manifolds*	January 1993
93–5	Fernando Viera and Roger Grimshaw	*Topographic forcing of coastal mesoscale phenomena: filamentation, vortex formation and eddy detachment*	January 1993
93–6	L. M. Shneerson and D. Easdown	*Growth and Existence of Identities in a Class of Finitely Presented Inverse Semigroups with Zero*	February 1993
93–7	N. Sabadini and R. F. C. Walters	*On Functions and Processors: an automata-theoretic approach to concurrency through distributive categories*	March 1993
93–8	N. Sabadini, S. Vigna and R. F. C. Walters	*An automata-theoretic approach to concurrency through distributive categories: on morphisms*	March 1993
93–9	Shu-Hao Sun and R. F. C. Walters	*Representations of modules and cauchy completeness*	March 1993