ISSN 1038-751X

Report 93-29 September 1993

On distributive automata and asynchronous circuits

N. Sabadini, R.F.C. Walters and Henry Weld

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

On distributive automata and asynchronous circuits

N. Sabadini,

Dipartimento di Scienze dell'Informazione,

Università di Milano, Via Comelico 39/41, I-20135 Milano MI, Italy

FAX: +39-2-55006276; e_mail: `sabadini@imiucca.csi.unimi.it`

R.F.C. Walters,   Henry Weld.

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

e_mail: `walters_b@maths.su.oz.au`, `weld_h@maths.su.oz.au`,

1. Introduction

In this paper we describe, using the calculus of distributive automata, a method for constructing from components *self-timed* circuits whose external behaviour is *insensitive to delays* in the components. This calculus, which we recall briefly below, was introduced by Sabadini and Walters in [SW93] and was based on ideas in [KW], [Wal91] and [Wal92]. We call these circuits *distributive circuits*. The main result is that any distributive automaton may be modelled by a distributive circuit. Following [SVWa], this means that if the state space of a wire is allowed to be $\mathbb{N}$ then all recursive functions can be computed by distributive circuits. There are two critical aspects to the proof:

(i) the encoding of sums in products [KW] which is needed since sums are allowed in the construction of automata but only products in the construction of circuits;

(ii) a synchronizing component is necessary to construct the product of two functional components.

The elementary categorical concepts used in this paper may be found in [Wal91].

This work has been supported by the Australian Research Council, Esprit BRA ASMICS, Italian MURST 40%, and the Italian CNR, contract 92.00529.CT01.

2. Distributive Automata

In the model of concurrency introduced in [SW93], sytems are represented by particular deterministic automata called distributive automata.

Distributive automata are automata constructed from a given family of sets and functions (data types and data operations) using the operations of a distributive category. That is, the alphabet and state space of a distributive automaton is formed by the operations of sum and product from some basic sets. The actions of a distributive automaton are formed from basic functions by composition, sum, and product of functions, projections, injections, the diagonal and codiagonal, and the distributive isomorphism $X \times (Y + Z) \cong X \times Y + X \times Z$.

For simplicity in this paper we consider alphabets with a single letter which we call *time*.

There is one further operation. A distributive automaton whose alphabet is one letter and whose state space is of the form $X + U + Y$ may compute by iteration a (total) function from X to Y; such automata we call pseudofunctions. In the construction of distributive automata we may use the function computed by a pseudofunction. This operation allows hiding of state, and encapsulation of iteration. Notice that the notion of pseudofunction has a precursor in Elgot's iteration theories [Elg75] and Heller's work on recursion categories [Hel90]. A similar definition can also be found in [Knu73], and [Mil71].

3. Distributive Circuits

We have in mind circuits made by putting together components using the usual operations available with wires - that is, using the operations of categorical products. We also have in mind continuous pulses travelling through the circuit. The precise physical assumptions will be explained in the full paper, but for simplicity in this abstract we give a model with discrete time (*not* however a *clock*) acting on the circuit.

First consider the *basic functional components* (bfc's) out of which a distributive circuit is constructed. Each bfc has a ribbon of input wires and a ribbon of output wires. The ith strand of a ribbon has a (not necessarily discrete) state space of the form $(X_i + 1)$, the summand 1 corresponding to *ground state*. The component also has an internal state space of the form $(U + 1)$. Hence the total state space of the component has the form

$$\prod_{i \text{ input}} (X_i + 1)\,(U+1) \prod_{j \text{ output}} (Y_j + 1) \cong (X+1)(U+1)(Y+1).$$

However only part of this state space is *permitted*. The permitted state space of a ribbon with state space $\prod_{i=1}^{n}(X_i + 1)$ is determined by a specific arithmetic expression containing each X_i exactly once.

Notice that any such arithmetic expression is canonically a subset of $\prod_{i=1}^{n}(X_i + 1)$. This coding of arithmetic expressions was introduced in [KW].

For example, a ribbon of three wires might have state space $(X_1 + 1)(X_2 + 1)(X_3 + 1)$ but permitted state space $X_1(X_2 + X_3)$. Using the underscore symbol (_) to denote ground state we have:

$$\begin{array}{rcl} X_1(X_2+X_3) & \subseteq & (X_1+1)(X_2+1)(X_3+1) \\ (x_1,(x_2,0)) & \mapsto & (x_1,x_2,\_) \\ (x_1,(x_3,1)) & \mapsto & (x_1,\_,x_3) \end{array}$$

The action of *time* on a bfc is to "compute a function", f say, from the arithmetic expression of the input ribbon to the arithmetic expression of the output ribbon over a period of time (dependent on the input state). That is, the behaviour is of the form

$$(x,\_,\_) \mapsto (\_,u_0,\_) \mapsto (\_,u_1,\_) \mapsto \cdots \mapsto (\_,\_,f(x)) \mapsto (\_,\_,f(x)) \mapsto \cdots$$

and

$$(\_,\_,\_) \mapsto (\_,\_,\_).$$

If a functional component has as input arithmetic expression A, output arithmetic expression B and computes function $f : A \to B$ we often denote it $f : A \,\longrightarrow\!\!\!\!\!+\!\!\!>\, B$. This idea arises from the notion of pseudofunction [KW], and from the idea of a family of pulses entering a component simultaneously, being processed, and the resulting family exiting the component simultaneously. Notice that the external behaviour of a bfc and of any circuit made from bfc's will be insensitive to internal delays inside the components.

Now a general distributive circuit is one constructed from bfc's by first forming larger functional components (fc's) and finally making a loop of an fc. An essential feature is that larger functional components will satisfy the same specifications as bfc's.

4. Constructions On Functional Components

4.1. *Composition of two fc's*

Consider fc's $f : A \longrightarrow\!\!\!\!\!\!\!+\;\; B$ with state space $(X+1)(U_f+1)(Y+1)$ and alphabet $time_f$, and $g : B \longrightarrow\!\!\!\!\!\!\!+\;\; C$ with state space $(Y+1)(U_g+1)(Z+1)$ and alphabet $time_g$. An fc $gf : A \longrightarrow\!\!\!\!\!\!\!+\;\; C$ which computes the composite function $g \circ f$ is obtained by placing the components f and g in series. This has state space $(X+1)(U_{gf}+1)(Z+1)$ and alphabet $time_{gf}$ and is constructed as follows. Let

$$(U_{gf}+1) = (U_f+1)(Y+1)(U_g+1)$$

and

$$time_{gf} : (X+1)(U_{gf}+1)(Z+1) \longrightarrow (X+1)(U_{gf}+1)(Z+1)$$

be the composition

$$(time_f \times (U_g+1)(Z+1))\ ((X+1)(U_f+1) \times time_g).$$

4.2. *Product of two fc's*

Given fc's $f : A \longrightarrow\!\!\!\!\!\!\!+\;\; B$ with state space $(X_f+1)(U_f+1)(Y_f+1)$ and alphabet $time_f$ and $g : C \longrightarrow\!\!\!\!\!\!\!+\;\; D$ with state space $(X_g+1)(U_g+1)(Y_g+1)$ and alphabet $time_g$, we construct a new fc $f \times g : A \times C \longrightarrow\!\!\!\!\!\!\!+\;\; B \times D$, with state space $(X_{(f\times g)}+1)(U_{(f\times g)}+1)(Y_{(f\times g)}+1)$ and alphabet $time_{(f\times g)}$, which computes the product of f and g as follows. We place the two components in parallel followed by a synchronizing device *SYNCH* which ensures that the product outputs simultaneously. The synchronising device *SYNCH* is <u>not</u> an fc. The synchronizer has state space of the form $(Y_f+1)(Y_g+1)S(Y_f+1)(Y_g+1)$. The device *SYNCH*, under the action of time, awaits arrival of a permitted input (not necessarily simultaneously) and then at some later time simultaneously outputs the same permitted data. The internal state S involves some storage facility.

The state spaces of the input ribbon, internal, and output ribbon of $f \times g$ are respectively

$$(X_{(f\times g)}+1) = (X_f+1)(X_g+1),$$

$$(U_{(f\times g)}+1) = (U_f+1)(U_g+1)(Y_f+1)(Y_g+1)S,$$

$$(Y_{(f\times g)}+1) = (Y_f+1)(Y_g+1).$$

Now, with k an appropriate permutation of the state space, $time_{f\times g}$ is given by the composition

$$k^{-1}\,(time_f \times time_g \times S(Y_f+1)(Y_g+1))\ k\,((X_f+1)(X_g+1)(U_f+1)(U_g+1) \times time_{SYNCH}).$$

Notice that while the two functions $(f \circ g) \times (h \circ k)$ and $(f \times h) \circ (g \times k)$ are the same they give rise to different constructions. The first involves only one synchroniser while the second involves two.

4.3. *Sum of two fc's*

Given two fc's f and g (as in 4.2), we construct an fc $f+g : A+C \longrightarrow\!\!\!\!\!\!\!+\;\; B+D$, with state space $(X_{(f+g)}+1)(U_{(f+g)}+1)(Y_{(f+g)}+1)$ and alphabet $time_{(f+g)}$, which computes the sum of f and g by placing the components in parallel.

The state spaces of the input ribbon, internal, and output ribbon of $f+g$ are respectively

$$(X_{(f+g)}+1) = (X_f+1)(X_g+1),$$

$$(U_{(f+g)}+1) = (U_f+1)(U_g+1),$$

$$(Y_{(f+g)}+1) = (Y_f+1)(Y_g+1).$$

and with k an appropriate permutation of the state space

$$time_{f+g} = k^{-1}\,(time_f \times time_g)\,k.$$

4.4. *Iteration and pseudofunctions*

A pseudofunction is an automaton whose alphabet is one letter and whose state space is of the form $X+U+Y$ which computes by iteration a (total) function from X to Y. Suppose a functional component $f : A+B+C \longrightarrow\!\!\!\!\!\!\!+\;\; A+B+C$ with state space $(X+1)(U+1)(Y+1)(U_f+1)(X+1)(U+1)(Y+1)$ and alphabet $time_f$ computes on iteration a function $\mathrm{call}(f) : A \rightarrow C$. We can construct a functional component $\mathrm{call}(f) : A \longrightarrow\!\!\!\!\!\!\!+\;\; C$ by feeding the output ribbon of f corresponding to U back into the input ribbon of f corresponding to U, and ignoring the input ribbon Y and output ribbon X, resulting in a state space for $\mathrm{call}(f)$ of

$$(X+1)(U+1)(Y+1)(U_f+1)(X+1)(U+1)(Y+1)$$

with

$$(X_{\mathrm{call}(f)}+1) = (X+1),$$

$$(U_{\mathrm{call}(f)}+1) = (U+1)(Y+1)(U_f+1)(X+1)(U+1),$$

$$(Y_{\mathrm{call}(f)}+1) = (Y+1)$$

The action of time on this fc is given by the composition

$$time_{\mathrm{call}(f)} = k^{-1}\,((X+1)\times twist \times (Y+1)(U_f+1)(X+1)(Y+1))\;k\;time_f$$

where k is a permutation bringing the $U+1$ terms together and *twist* is on the $U+1$ terms.

4.5. *Special functional components*

The injection and projection functions, diagonal and codiagonal functions are easily implemented with wires. For example diagonal is splitting of wires, and codiagonal is merging (appropriate because we are considering only permitted states).

However we need a special basic functional component for each distributive law.

4.6. *The Theorem*

Consider a graph G of basic sets and functions for constructing distributive automata. Suppose we have bfc's corresponding to each of the functions in G, and suppose we have synchronizers and distributive law components, then we can construct a distributive circuit whose behaviour (ignoring internal states of the bfc's) is the same as the distributive automaton.

The precise sense in which it is the same can be described in terms of the notion of refinement [SVWb].

5. Comparisons

In the full paper we will make comparisons between our methods and other methods of constructing self-timed and delay-insensitive circuits (for example, [Ebe91], [YGD92], [SG88], [Mar90], [Mar86], [Sut89], [OSC67], [RMTF88], [Udd86], [JU90], [JU91]).

References

[Ebe91] J.C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3), 1991.

[Elg75] C. Elgot. Monadic computation and iterative algebraic theories. *Studies in Logic and the Foundations of Mathematics*, 80:175–230, 1975.

[Hel90] A. Heller. An existence theorem for recursion categories. *Journal of Symbolic Logic*, 55(3):1252–1268, 1990.

[JU90] M.B. Josephs and J.T. Udding. Delay-insensitive circuits: An algebraic approach to their design. In *CONCUR '90. Theories of Concurrency. Unification and Extension*, 1990.

[JU91] M.B. Josephs and J.T. Udding. An algebra for delay-insensitive circuits. In E.M. Clark and R.P. Kurshan, editors, *Computer Aided Verificartion '90*, number 30 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1991.

[Knu73] D.E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1973.

[KW] W. Khalil and R.F.C. Walters. An imperative language based on distributive categories II. *RAIRO Informatique Théorique et Applications*. To appear.

[Mar86] A.J. Martin. Compiling communicating processes into delay-insensitive circuits. *Distributed Computing*, 1:226–234, 1986.

[Mar90] A.J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C.A. Hoare, editor, *Developments in Concurrency and Communication*. 1990.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. of the 2nd Joint Conference on Artificial Intelligence*, pages 481–489. BCS, 1971.

[OSC67] S.M. Ornstein, M.J. Stucki, and W.A. Clark. Macromodular computer systems. Number 30 in AFIPS Spring Joint Computing Conference, 1967.

[RMTF88] F.U. Rosenberger, C.E. Molnar, T.J.Chaney, and T. Fang. Q-modules: Internally clocked, delay insensitive modules. *IEEE Transactions On Computers*, 37(9):1005–1018, September 1988.

[SG88] J. Staunstrup and M.R. Greenstreet. From high-level descriptions to VLSI circuits. *BIT*, 28(3):620–638, 1988.

THE UNIVERSITY OF SYDNEY

Research Reports of the School of Mathematics and Statistics[†]

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
93–10	Rowan Kemp	*Tableaux and Rank-selection in Fibonacci Lattices*	March 1993
93–11	K. F. Lai and Kam Ping Mok	*On the Differential Geometry of the* $(1,1)$ *Tensor Bundle*	April 1993
93–12	A. M. N. Fu	*Statistical Analysis of an Autoassociative Memory Network*	April 1993
93–13	Shu-Hao Sun	*Generalized Grothendieck topologies*	April 1993
93–14	Michael S. Farber and Jerome P. Levine	*Deformations of the Atiyah-Parodi-Singer Eta-invariant*	May 1993
93–15	Y. H. Harris Kwong and D. G. Rogers	*A matrix method for counting Hamiltonian cycles on grid graphs*	May 1993
93–16	G. Janelidze and G. M. Kelly	*Galois Theory and a General Notion of Central Extension*	May 1993
93–17	Deryn Griffiths	*Series Expansions of Algebraic Functions*	June 1993
93–18	Roman R. Poznanski and Loyd L. Glenn	*Estimating the Effective Electrotonic Length of Dendritic Neurons with Reduced Equivalent Cable Models*	June 1993
93–19	A. Carboni, G. M. Kelly and M. C. Pedicchio	*Some remarks on Maltsev and Goursat categories*	June 1993
93–20	Jonathan A. Hillman	*Free products and 4-dimensional connected sums*	July 1993
93–21	Fernando Viera	*Boundary surface dynamics: An algorithm for stratified geostrophic flows*	July 1993
93–22	Jonathan A. Hillman	*Embedding homology equivalent 3-manifolds in 4-space*	August 1993
93–23	Laurentiu Paunescu	*Simple equations for some four-dimensional manifolds*	August 1993
93–24	G. I. Lehrer	*The space of invariant functions on a finite Lie algebra*	August 1993
93–25	Lars Thams	*The subcomodule structure of the symmetric powers of the natural module of quantum* GL_n.	August 1993
93–26	R. Graham and H. Luckock	*Nicolai maps for quantum cosmology*	August 1993
93–27	H. Luckock	*Supersymmetric quantum cosmology*	September 1993
93–28	N. Sabadini, S. Vigna and R.F.C. Walters	*A notion of refinement for automata*	September 1993
93–29	N. Sabadini, R.F.C. Walters and Henry Weld	*On distributive automata and asynchronous circuits*	September 1993

[†]Editor: Dr D. E. Taylor