# AN ALGEBRA OF MULTIDIMENSIONAL ARRAYS [*]

GAÉTAN HAINS AND LENORE M. R. MULLIN [†]

**Abstract.** We present a construction of the MOA algebra of arrays and its calculus of functions on multidimensional arrays. Arrays are treated as a single data type in any number of dimensions, unlike previous definitions with nested lists, preventing the occurrence of an explosive number of algebraic laws. The concepts of scalar and empty array are explained and provide uniform arguments and values for the dimension-independent operators. All operations are total and error conditions (e.g. bounds overflows) are enforced by the absorbent property of the empty array.

Because arrays are represented as pairs of lists, an MOA expression has a meaning as a strict functional program. Because arrays have forms, an MOA expression implicitly defines communication patterns for its parallel evaluation. We propose that this calculus constitutes a basis for functional programming with arrays on distributed-memory architectures having array-like topologies: linear, meshes and hypercubes. The placement of arrays on such architectures can be encoded by functions of the array forms. Array forms also allow parallel operators to have definitions which factor communication (form dependent) from computation (content dependent). With fixed placement rules, known parallel dataflows for each array operator, it will be possible to predict statically a realistic complexity for the parallel program corresponding to a given MOA expression. Identities of the array algebra can then be used as optimisations when applied in the direction of decreasing complexity.

**1. Introduction.** We present a construction of the MOA algebra of arrays [24] as a heterogeneous algebra of functions on multidimensional arrays, lists and array forms. Functions are strongly typed and curried so function application associates to the left, unless indicated otherwise. Simple function arguments are written without parentheses. To simplify notation, we sometimes avoid fully parenthesised expressions where context or types can resolve ambiguity. Function composition of $g$ and $f$ is written $f.g$:

$$f.g\,x = f(gx).$$

Arrays are a single type in any number of dimensions, unlike previous definitions as homogeneous nested lists. The content of an array is treated separately from its structure called shape or *form*. This simplifies global operations and prevents the occurrence of an explosive number of algebraic laws by making them independent of the number of dimensions.

All our operations are total over their list/array domains but some return the empty array for non-empty arguments. In particular we do not use functions undefined for the empty list or the empty array. Because arrays are essentially defined as pairs of lists, an MOA expression is a strict functional program on pairs and lists, which gives it a unique meaning. We therefore avoid the definition of a separate semantics for MOA. The form components however suggest regular communication patterns by which the functional program may be implemented.

**2. Why a Calculus of Arrays.** The definitions presented here and their order of dependencies are not random. They are meant to constitute more than a functional language in abstract syntax. Our main goal is to create a framework for proving and classifying geometric properties of data and algorithms, with enough precision for proofs to be automatically verifiable. Computational properties of arrays can be stated precisely, proved formally

[*] First author supported by an NSERC operating grant

[†] G. Hains, Informatique et Recherche Opérationnelle, Université de Montréal, C.P. 6128, succursale A, Montréal, Québec, Canada H3C 3J7, hains@iro.umontreal.ca. L. M. R. Mullin, Computer Science and Electrical Engineering, University of Vermont, Burlington, Vermont, USA 05405, lenore@newton.uvm.edu

and used safely by compilers or interpreters. Proofs then cover limit cases (empty blocks, unequal dimensions etc.) which may cause incorrect implementations if treated informally.

The result of this methodology will be safer yet more complex structure-related optimisations, reducing dependence on costly run-time support of routing. We thus advocate a multidimensional extension of existing vectorising compiler methods which analyse architecture topology and program "loop" structures. Compilers can be understood as transformers from programs (functions on arrays) to process networks (arrays of functions) and forms provide information for improving communication locality and load balancing within the array of functions.

The current notion of bulk-synchronous computation [33, 34] takes the opposite view that processor networks should be made to emulate shared memory in a universal way, avoiding static optimisations. This approach to parallel programming hides locality by distributed randomised routing. It is our belief that randomised routing will not provide the highest performances possible with parallel architectures. Networks of large bandwidth such as hypercubes are known to require long communication delays in the large scale [35], and low-dimensional meshes which are scalable have insufficient bandwidth for efficient bulk-synchronised computation [29]. These facts suggest that efficient placement of data structures on parallel architectures will remain a basic concern in the design of compilers and languages for high-performance.

**3. Why Non-Recursive Arrays.** Multidimensional arrays provide an appropriate structure for many data-parallel programs and the need for arrays is generally accepted in scientific supercomputing. Arrays also model spatial regularity in a large class of applications and in many distributed-memory architectures: pipelines, meshes, hypercubes. If we are to use arrays to model both data and mapping to processes/machines, then operators which are independent of the number of dimensions should be preferred: mapping the same program to a two-dimensional mesh or to a hypercube should be expressed as the same operation with different parameters. The MOA formalism describes arrays in a uniform dimension-independent algebra.

An advantage of having explicit forms is that, in the restricted context of arrays, the formulation of program mapping to parallel architectures as an NP-complete graph embedding problem may be avoided. When statically mapping arrays to meshes, it is sufficient to consider the respective forms of data and architecture (e.g. a hypercube is a $[2, 2, \ldots, 2]$-mesh) [30, 31] by using generalised Gray codes.

It is possible to define arrays as rectangular nested lists [5], the *partial* free algebra generated from singleton arrays by array concatenation. One problem with this definition is that there are then an infinite number of concatenation operators (left, right, top, bottom, up, down , etc.) so that algebraic identities come in large numbers. MOA laws are stated and applied independently of the number of dimensions, arrays being non-recursive.

The use of non-recursive arrays may also prevent equational term matching of nested lists or nested multisets at execution time. If a functional programming system is to support global and deterministic optimisations, it must rely on the shape of data structures. This suggests the use of non-recursive arrays with explicit shape information (which we advocate) or equational term matching of recursive shape patterns with recursive array terms. Matching is efficient for one-dimensional arrays or lists but becomes prohibitive if multi-dimensional objects are processed as nested terms. For example Banâtre et al. [2] study parallel declarative programming where the elementary run-time operation is to search a multiset for matching terms. This design avoids the cause of complexity in associative-commutative matching (TM-AC), namely nesting. If arrays were implemented as nested multisets on the abstract machine of Banâtre et al., the matching problem would become NP-complete [3]. Jayaraman and Plaisted [15] discuss sequential functional programming with sets and the use of associative-commutative term matching (TM-AC). A key remark concerning efficiency is that

> Although the complexity of [TM-AC] in general is NP-complete [3], most SEL patterns in practice have very simple structure, with non-repeating variables [linear terms], and hence can be matched reasonably fast.

In this other system, the previous source of complexity is avoided (TM-AC is polynomial for linear terms) but still could not support fast parallel execution: there is evidence that TM-AC for linear terms may not be in the complexity class NC (the problem of perfect matching in bipartite graphs BPM, is logspace-reducible to TM-AC for linear terms, and BPM is not known to be in NC [12, 18]. The above reasoning assumed that lists would be implemented by associative-commutative terms i.e. multisets, but nested *lists* would have the same dynamic costs or worse, if associative, non-commutative term matching was used [27, 28]. In agreement with our conclusions, Hudak observes scheduling and synchronisation overheads for recursively defined arrays in functional languages [14].

An early implementation of arrays was for Abrams' APL machine [1]. It used a technique to avoid data movements in array transposition through the exchange of index pointers. In view of the above remarks about efficiency, we believe that such optimisations based on the analysis of array forms are central to the efficiency of parallel programming systems.

The next section defines array forms as equivalence classes of lists of natural numbers. In the rest of the text, proposition and theorem statements should be read as universally quantified over the most general types of their free variables.

**4. Lists and Forms.** Let $T^*$ be the set of lists of elements from set $T$, the free monoid generated from $T$ by the singleton list constructor $[\_] : T \to T^* \ : \ t \mapsto [t]$, the empty list $[\,]$ and concatenation $+\!\!+ : T^* \to T^* \to T^*$ [5]. Let $\# : T^* \to \mathbf{N}$ return the length of a list and $\Pi : \mathbf{N}^* \to \mathbf{N}$ the product of a list of naturals $(\Pi\,[\,] = 1)$. Let $\equiv$ be the finest equivalence relation on $\mathbf{N}^*$ which is a congruence for concatenation and such that $\forall n \in \mathbf{N}$. $[0,n] \equiv [n,0] \equiv [0]$ and $[1] \equiv [\,]$. Modulo $\equiv$, occurrences of 0 are absorbent and occurrences of 1 are absorbed. The implications of this definition are explained in section 5.

Definition 1.

$$\begin{aligned}(\equiv) = \ \bigcap\{E \subseteq \mathbf{N}^* \times \mathbf{N}^* \mid \quad & E \textit{ is an equivalence relation,}\\ & x_1\,E\,y_1, x_2\,E\,y_2, \ \Rightarrow \ (x_1 +\!\!+ x_2)\,E\,(y_1 +\!\!+ y_2),\\ & [0,n]\,E\,[n,0]\,E\,[0], \quad [1]\,E\,[\,] \quad \}\end{aligned}$$

*The equations* $[0,n] \equiv [n,0] \equiv [0]$ *and* $[1] \equiv [\,]$ *will be called the* defining equations *for* $\equiv$.

Proposition 4.1. $l_1 \equiv l_2 \Rightarrow \Pi\,l_1 = \Pi\,l_2$.

*Proof.* Let $\equiv'$ be the kernel relation for $\Pi$. Then $\equiv'$ satisfies the defining equations for $\equiv$ and is a congruence for concatenation ($\Pi$ is associative). By definition $\equiv$ is therefore a refinement of $\equiv'$. □

Lemma 4.2. $\forall l, l' \in \mathbf{N}^*$. $l \equiv l'$ *if and only if* $\exists l''$ *which can be obtained from* $l$ *and* $l'$ *by a finite number of applications of the following rewrite rules:*

$$\begin{array}{rrcl} (\mathbf{1}) & [0,n] & \to & [0] \\ (\mathbf{2}) & [n,0] & \to & [0] \\ (\mathbf{3}) & [1] & \to & [\,] \end{array}$$

*Proof.* The set of rules is Noetherian (no infinite application chains possible) because each rule shortens lists, and confluent (independent of application order). Confluence is a consequence of the following observation: any list containing 0 will be rewritten to [0], and any list containing 1 will have its occurrences removed by the third rule. The rewriting algorithm therefore assigns to any list of naturals a unique normal form. The associated equivalence relation (the kernel of the map $l \mapsto \text{NormalForm}(l)$) preserves the defining equations of $\equiv$, is a congruence for concatenation and so $\equiv$ is a refinement of it. Conversely, if a congruence for concatenation preserves the defining equations, it must contain the kernel relation of the rewriting system. □

PROPOSITION 4.3. *Every class of $\equiv$ contains a unique list of minimal length.*

*Proof.* From the proof of lemma 4.2, a list of naturals in normal form is of minimal length. □

DEFINITION 2. *The set of* forms $\mathcal{F}$ *is* $\mathbf{N}^*/\equiv$.

COROLLARY 4.4. $\mathcal{F} \simeq \{[0]\} \cup \mathbf{N}_2^*$ *where* $\mathbf{N}_2 = \mathbf{N} - \{0,1\}$.

It follows from definition 1, proposition 4.1 and definition 2 that the empty form, concatenation of forms and multiplication of forms are well defined operations. The length of a form however depends on its normal form (see definition 8).

**5. Arrays.** We now define arrays as forms equipped with contents of conformable length.

DEFINITION 3. *The set of* arrays *containing elements from set $T$ is defined as*

$$T^{\Box} = \{(f,c) \in \mathcal{F} \times T^* \mid \Pi f = \#c\}.$$

*The first component of an array is called its* form *or shape and the second its* content.

The content of an array is understood to be in lexicographic order of indexes. This notion will be made precise by the definition of an indexing operator. Because 0 is absorbent in forms, there is a unique array with empty content: $\Theta$. For any set $T$, empty or not, $\Theta \in T^{\Box}$.

DEFINITION 4. $\Theta = ([0],[\,])$ *is called the* empty array.

Arrays with empty form must have singleton contents and are called *scalars*. Conversely, arrays with singleton contents must be scalars.

DEFINITION 5. *The scalar constructor is* `scl` $: T \to T^{\Box} : t \mapsto ([\,],[t])$.

DEFINITION 6. *So-called* vectors *(though they don't have to constitute a vector space) are constructed by*

$$\texttt{vec} : T^* \to T^{\Box} : c \mapsto (\mathit{NormalForm}[\#c], c)$$

It follows that `scl` $a =$ `vec` $[a]$ and the set of scalars is included in that of vectors. Also, $\Theta$ is a non-scalar vector: `vec` $[\,]$.

DEFINITION 7. *The* shape *operator returns the form of an array*

$$\rho : T^{\Box} \to \mathcal{F} : (f,c) \mapsto f$$

*the* content *operator returns its content*

$$\texttt{snd} : T^{\Box} \to T^* : (f,c) \mapsto c$$

*and the* size *operator returns the length of the content i.e. the product of the shape*

$$\texttt{siz}\ : T^{\square} \to \mathbf{N} : (f,c) \mapsto \Pi f.$$

The fact that occurrences of 1 are absorbed in forms allows arrays to have a dimension which is geometrically unique, avoiding for example notions of column or row vectors.

DEFINITION 8. *Let rep* : $\mathbf{N}^* \to \mathbf{N}^* : l \mapsto \mathit{NormalForm}(l)$ *as defined in the proof of lemma 4.2. Then* $\#.\mathit{rep}$ *is well defined modulo* $\equiv$ *and will provide a dimension operator for arrays. The* dimension *operator returns the length of the shortest representative list for the form*

$$\texttt{dim}\ : T^{\square} \to \mathbf{N} : (f,c) \mapsto \#(\mathit{rep}\ f).$$

Scalars are therefore zero-dimensional and other vectors are one-dimensional. An unexpected consequence of the definition of `dim` is that $\texttt{dim}\,\Theta = 1$, the empty array is not a scalar. In fact, the empty array often plays the role of an undefined array [1] and there is no external reason why it should be one-dimensional.

We now define the list ordinal constructor which will be used later in generating index lists.

DEFINITION 9. $\iota : \mathbf{N} \to \mathbf{N}^* : \iota 0 = [\,] : \iota(n+1) = \iota n +\!\!+ [n]$.

Basic properties of the empty array, scalars and vectors, for $t \in T$ and $l \in T^*$:

$$\begin{array}{ccc} \texttt{dim}\,\Theta = 1 & \texttt{dim}\,(\,\texttt{scl}\ t) = 0 & \texttt{dim}\,(\,\texttt{vec}\ l) \le 1 \\ \texttt{siz}\,\Theta = 0 & \texttt{siz}\,(\,\texttt{scl}\ t) = 1 & \texttt{siz}\,(\,\texttt{vec}\ l) = \#l \\ \rho\,\Theta = [0] & \rho\,(\,\texttt{scl}\ t) = [\,] & \rho\,(\,\texttt{vec}\ l) = \text{NormalForm}[\#l] \end{array}$$

**6. Restructuring.** The `rav` operator flattens an array into a vector having the same content.

DEFINITION 10. $\texttt{rav}\ : T^{\square} \to T^{\square} : (f,c) \mapsto \texttt{vec}\ c$. In particular, $\texttt{rav}\,(\,\texttt{scl}\ t) = \texttt{scl}\,t$ and $\texttt{rav}\,\Theta = \Theta$.

The *reshape* operator $\widehat{\rho} : \mathcal{F} \to T^{\square} \to T^{\square}$ restructures an array to a new form. In the case of a non-conformable form it returns the empty array.

DEFINITION 11. $\widehat{\rho}\,f\,A = (\Pi f = \texttt{siz}\ A \to (f,\ \texttt{snd}\ A);\ \Theta)$ *where* $(p \to A;\ B)$ *abbreviates* **if** $p$ **then** $A$ **else** $B$.

By definition $\widehat{\rho}\,fA$ is always an array and so $\widehat{\rho}$ is well defined. In general $\widehat{\rho}\,f(\,\widehat{\rho}\,gA) \neq \widehat{\rho}\,fA$ because $\widehat{\rho}\,gA$ may be $\Theta$. For example if $A = \texttt{vec}\ (\iota 4)$ then

$$\widehat{\rho}\,[2,2]A = ([2,2],\iota 4) \neq \widehat{\rho}\,[2,2](\,\widehat{\rho}\,[5]A) = \widehat{\rho}\,[2,2]\Theta = \Theta.$$

Flattening is a special case of reshaping and is invertible.

PROPOSITION 6.1. $\texttt{rav}\,A = \widehat{\rho}\,[\,\texttt{siz}\ A]A$ *and* $\widehat{\rho}\,(\rho\,A)(\,\texttt{rav}\,A) = A$.

*Proof.*

$$\begin{array}{rcl} \widehat{\rho}\,[\,\texttt{siz}\ A]A & = & (\Pi\,[\,\texttt{siz}\ A] = \texttt{siz}\ A \to ([\,\texttt{siz}\ A],\ \texttt{snd}\ A);\ \Theta) \\ & = & (\texttt{true} \to ([\,\texttt{siz}\ A],\ \texttt{snd}\ A);\ \Theta) \end{array}$$

[1] for 'multiplicative' operations like $\oplus$ it is absorbent but for 'additive' operations like `stk` it is a zero element

$$\begin{aligned}
&= ([\,\texttt{siz}\,A], \texttt{snd}\,A) \;=\; ([\#\,\texttt{snd}\,A], \texttt{snd}\,A)\\
&= \texttt{vec}\,.\,\texttt{snd}\,A \;=\; \texttt{rav}\,A.\\
\widehat{\rho}\,(\rho\,A)(\,\texttt{rav}\,A) &= (\Pi\,\rho\,A = \texttt{siz}\,.\,\texttt{rav}\,A \rightarrow (\rho\,A,\ \texttt{snd}\,.\,\texttt{rav}\,A);\ \Theta)\\
&= (\,\texttt{siz}\,A = \texttt{siz}\,.\,\texttt{rav}\,A \rightarrow (\rho\,A,\ \texttt{snd}\,A);\ \Theta)\\
&= (\rho\,A,\ \texttt{snd}\,A) = A.
\end{aligned}$$

□

**7. Array Types.** With the goal of automatic verification in mind, it is necessary to formally justify writing arrays as pairs of lists. An array type is not a product but a dependent type. Since arrays from $T^{\square}$ are not merely elements of $\mathbf{N}^* \times T^*$, it would be necessary to label forms and arrays by type constructors, which we have not done for clarity. To avoid any possible confusion between arrays and pairs, phrases of the kind "the array $(l, c)$" should be understood to mean "the array $\widehat{\rho}\,f(\,\texttt{vec}\ c)$" where $f$ is the form with list representative rep $l$. In other words, $([1,1],[a])$ will mean $\texttt{scl}\ a$, $([0,4],[\,])$ will mean $\Theta$, and $([2],[a,b,c])$ would also mean $\Theta$, although this last phrase is of no practical use. A similar remark applies to isolated forms written as lists of integers.

**8. Indexing.** Direct access is a key feature of concrete arrays, and this is modeled by an indexing operator. The *take* and *drop* list operations are needed in its definition.

Definition 12.

$$\begin{array}{ll}
\texttt{tk}\ : \mathbf{N} \rightarrow T^* \rightarrow T^* & \texttt{dr}\ : \mathbf{N} \rightarrow T^* \rightarrow T^*\\
\texttt{tk}\,0\,l = [\,] & \texttt{dr}\,0\,l = l\\
\texttt{tk}\,(n+1)[\,] = [\,] & \texttt{dr}\,(n+1)[\,] = [\,]\\
\texttt{tk}\,(n+1)([x] +\!\!+ l) = [x] +\!\!+ \texttt{tk}\,n\ l & \texttt{dr}\,(n+1)([x] +\!\!+ l) = \ \texttt{dr}\,n\ l
\end{array}$$

Notice that both are total functions, unlike for example the `head` function: $n > \#l \Rightarrow \texttt{tk}\,n\,l = l$ and $\texttt{dr}\,n\,l = [\,]$. A direct induction proof on the above definitions shows that take and drop are related by

Proposition 8.1. $(\,\texttt{tk}\,n\,l) +\!\!+ (\,\texttt{dr}\,n\,l) = l.$

Definition 13. *Cardinal subtraction of integers is defined by*

$$x \dot{-} y = (x > y \rightarrow x - y;\ 0) = (x - y)\max 0$$

The following properties of `tk` and `dr` will be used in proving properties of the indexing operator $\Psi$.

Proposition 8.2. $(\,\texttt{dr}\,a).(\,\texttt{tk}\,b) = (\,\texttt{tk}\,(b \dot{-} a)).(\,\texttt{dr}\,a)$ *where* $b \dot{-} a = ((b-a)\max 0)$.

*Proof.*

$$\texttt{dr}\,a(\,\texttt{tk}\,0l) \;=\; \texttt{dr}\,a[\,] \;=\; [\,] \;=\; \texttt{tk}\,(0 \dot{-} a)[\,] \;=\; \texttt{tk}\,(0 \dot{-} a)(\,\texttt{dr}\,a[\,])$$

$$\begin{aligned}
\texttt{dr}\,a(\,\texttt{tk}\,(n+1)[\,]) &= \texttt{dr}\,a[\,]\\
&= [\,]\\
&= \texttt{tk}\,(n+1 \dot{-} a)[\,]\\
&= \texttt{tk}\,(n+1 \dot{-} a)(\,\texttt{dr}\,a[\,]).
\end{aligned}$$

$$\begin{aligned}\texttt{dr}\,0(\,\texttt{tk}\,nl) &= \texttt{tk}\,nl\\ &= \texttt{tk}\,(n\dot{-}0)l\\ &= \texttt{tk}\,(n\dot{-}0)(\,\texttt{dr}\,0l).\end{aligned}$$

$$\begin{aligned}\texttt{dr}\,(m+1)(\,\texttt{tk}\,(n+1)([x]\mathbin{+\!\!+}l)) &= \texttt{dr}\,(m+1)([x]\mathbin{+\!\!+}(\,\texttt{tk}\,nl))\\ &= \texttt{dr}\,m(\,\texttt{tk}\,nl)\\ &= \texttt{tk}\,(n\dot{-}m)(\,\texttt{dr}\,ml)\ \ \text{induction}\\ &= \texttt{tk}\,((n+1)\dot{-}(m+1))(\,\texttt{dr}\,(m+1)([x]\mathbin{+\!\!+}l)).\end{aligned}$$

□

COROLLARY 8.3. $(\,\texttt{tk}\,a).(\,\texttt{dr}\,b) = (\,\texttt{dr}\,b).(\,\texttt{tk}\,(a+b))$.
*Proof.* $(\,\texttt{tk}\,a).(\,\texttt{dr}\,b) = (\,\texttt{tk}\,((a+b)\dot{-}b)).(\,\texttt{dr}\,b) = (\,\texttt{dr}\,a).(\,\texttt{tk}\,(a+b))$. □
PROPOSITION 8.4. $(\,\texttt{tk}\,a).(\,\texttt{tk}\,b) = \texttt{tk}\,(a \min b) = (\,\texttt{tk}\,b).(\,\texttt{tk}\,a)$.
*Proof.*

$$\texttt{tk}\,0(\,\texttt{tk}\,bl) = [\,] = \texttt{tk}\,0[\,] = \texttt{tk}\,(0\min b)[\,]$$

$$\texttt{tk}\,a(\,\texttt{tk}\,0l) = \texttt{tk}\,a[\,] = [\,] = \texttt{tk}\,0[\,] = \texttt{tk}\,(a\min 0)[\,]$$

$$\texttt{tk}\,(m+1)(\,\texttt{tk}\,(n+1)[\,]) = \texttt{tk}\,(m+1)[\,] = [\,] = \texttt{tk}\,((m+1)\min(n+1))[\,]$$

$$\begin{aligned}\texttt{tk}\,(m+1)(\,\texttt{tk}\,(n+1)([x]\mathbin{+\!\!+}l)) &= \texttt{tk}\,(m+1)([x]\mathbin{+\!\!+}(\,\texttt{tk}\,nl))\\ &= [x]\mathbin{+\!\!+}(\,\texttt{tk}\,m(\,\texttt{tk}\,nl))\\ &= [x]\mathbin{+\!\!+}(\,\texttt{tk}\,(m\min n)l)\ \ \text{induction}\\ &= [x]\mathbin{+\!\!+}(\,\texttt{tk}\,(((m+1)\min(n+1))-1)l)\\ &= \texttt{tk}\,((m+1)\min(n+1))([x]\mathbin{+\!\!+}l).\end{aligned}$$

□

PROPOSITION 8.5. $(\,\texttt{dr}\,a).(\,\texttt{dr}\,b) = \texttt{dr}\,(a+b) = (\,\texttt{dr}\,b).(\,\texttt{dr}\,a)$.
*Proof.*

$$\texttt{dr}\,a(\,\texttt{dr}\,0l) = \texttt{dr}\,al = \texttt{dr}\,(a+0)l$$

$$\texttt{dr}\,a(\,\texttt{dr}\,(b+1)[\,]) = \texttt{dr}\,a[\,] = [\,] = \texttt{dr}\,(a+b+1)[\,]$$

$$\begin{aligned}\texttt{dr}\,a(\,\texttt{dr}\,(b+1)([x]\mathbin{+\!\!+}l)) &= \texttt{dr}\,a(\,\texttt{dr}\,bl)\\ &= \texttt{dr}\,(a+b)l\ \ \text{induction}\\ &= \texttt{dr}\,(a+b+1)([x]\mathbin{+\!\!+}l).\end{aligned}$$

□

Array *indexing* is expressed by an infix operator $\Psi : \mathbf{N}^* \to T^\Box \to T^\Box$ which is usually written as right-associative. The definition of $\Psi$ generalises standard usage such that when $\texttt{dim}\,A = 2$, $[x_0, x_1]\,\Psi\,A$ is a scalar and $[x_0]\,\Psi\,A$ is a vector slice of A. For example if $\rho\,A = [8,7]$, $[x_0, x_1]\,\Psi\,A$ will identify a scalar containing element $7x_0+x_1$ in $\texttt{snd}\,A$ (numbered from

0) and $[x_0]\,\Psi\,A$ will be a vector made from the elements numbered $7x_0, 7x_0+1, \ldots, 7x_0+6$. To make algebraic properties independent of the length of index lists, we also want to extend $\Psi$ to the case of empty or long indexes. The required behaviour is obtained by a careful combination of the properties of `tk` and `dr`.

DEFINITION 14. *The indexing operator* $\Psi : \mathbf{N}^* \to T^\square \to T^\square$ *is defined by induction on the first argument.*

- $[\,]\,\Psi\,A = A$
- $([i] + \!\!+ l)\,\Psi\,A = l\,\Psi\,([i]\,\Psi\,A)$.
  *where* $[i]\,\Psi\,A = (\,\widehat{\rho}\,t).\,\texttt{vec}\,.(\,\texttt{tk}\,\Pi\,t).(\,\texttt{dr}\,(i \cdot \Pi\,t)).\,\texttt{snd}\,A$
  *where* $t = \,\texttt{dr}\,1(\rho\,A)$.

LEMMA 8.6. $\Psi$ *is well defined,*

$$\forall l \in \mathbf{N}^* \;\; \forall A \in T^\square \quad l\,\Psi\,A \in T^\square.$$

*Proof.* By induction on $l$, following the definition of $\Psi$ and using the fact that both `vec` and $\widehat{\rho}$ are well defined. □

$\Psi$ returns the empty array when $l$ is out of bounds. Otherwise we will describe the array as being *correctly indexed.* When $\#l = \texttt{dim}\,A$ and all $l$'s entries are within the bounds set by $\rho\,A$, $l\,\Psi\,A$ returns the scalar from $A$'s content located at coordinates $l$. $l\,\Psi\,A$ returns subarrays of increasing dimensions with decreasing $\#l$ . For example if $A = ([2,2],[a,b,c,d])$ then

$$\begin{aligned}
[1]\,\Psi\,A &= (\,\widehat{\rho}\,[2]).\,\texttt{vec}\,.(\,\texttt{tk}\,2).(\,\texttt{dr}\,2)[a,b,c,d] \\
&= (\,\widehat{\rho}\,[2]).\,\texttt{vec}\,[c,d] \\
&= ([2],[c,d])
\end{aligned}$$

the second row of $A$. Also, the joint behaviour of `tk` and `dr` ensures that $\Psi$ returns $\Theta$ when out of bounds:

$$\begin{aligned}
[3]\,\Psi\,A &= (\,\widehat{\rho}\,[2]).\,\texttt{vec}\,.(\,\texttt{tk}\,2).(\,\texttt{dr}\,6)[a,b,c,d] \\
&= (\,\widehat{\rho}\,[2]).\,\texttt{vec}\,[\,] \\
&= \widehat{\rho}\,[2]\Theta = \Theta
\end{aligned}$$

i.e. there is no fourth row in $A$.

The following proposition expresses the fact that index lists are read from left to right.

PROPOSITION 8.7. $(r + \!\!+ l)\,\Psi\,A = l\,\Psi\,r\,\Psi\,A$.

*Proof.* By induction on $r$. Base case:

$$([\,] + \!\!+ l)\,\Psi\,A = l\,\Psi\,A = l\,\Psi\,[\,]\,\Psi\,A$$

Step:

$$\begin{aligned}
([x] + \!\!+ r + \!\!+ l)\,\Psi\,A &= ([x] + \!\!+ (r + \!\!+ l))\,\Psi\,A \\
&= (r + \!\!+ l)\,\Psi\,[x]\,\Psi\,A \\
&= l\,\Psi\,r\,\Psi\,[x]\,\Psi\,A \;\text{ by induction hyp.} \\
&= l\,\Psi\,([x] + \!\!+ r)\,\Psi\,A
\end{aligned}$$

□

The following property of $\Psi$ is not intuitive but is a consequence of its definition for all lists of naturals. The only correct index for a scalar is $[0]$. Otherwise, $\Psi$ returns the empty array. An array can thus be correctly indexed by an arbitrarily long list with suffix in $\{0\}^*$. For example if $A = ([2,2],[a,b,c,d])$ then

$$[1,0]\,\Psi\,A = [1,0,0]\,\Psi\,A = [1,0,0,0]\,\Psi\,A = \ldots = \texttt{scl}\;c$$

but $[1,0,1]\,\Psi\,A = \Theta$.

LEMMA 8.8. $[0]\,\Psi\,(\texttt{scl}\;a) = \texttt{scl}\;a$ *and* $\forall n \in \mathbf{N}.\;\; [n+1]\,\Psi\,(\texttt{scl}\;a) = \Theta$.

*Proof.*

$$\begin{array}{rcl}
[i]\,\Psi\;\texttt{scl}\;a & = & [i]\,\Psi\,([\,],[a]) \\
& = & (\widehat{\rho}[\,]).\texttt{vec}\,.\texttt{tk}\,(\Pi[\,]).\texttt{dr}\,(i\cdot\Pi[\,])[a] \\
& = & (\widehat{\rho}[\,]).\texttt{vec}\,.(\texttt{tk}\,1)(i=0\rightarrow[a];\;[\,]) \\
& = & (\widehat{\rho}[\,]).\texttt{vec}\,(i=0\rightarrow[a];\;[\,]) \\
& = & (\widehat{\rho}[\,])(i=0\rightarrow\texttt{vec}\,[a];\;\Theta) \\
& = & (i=0\rightarrow\widehat{\rho}[\,](\texttt{vec}\,[a]);\;\widehat{\rho}[\,]\Theta) \\
& = & (i=0\rightarrow\texttt{scl}\;a;\;\Theta)
\end{array}$$

□

The empty array is absorbent for indexing.

LEMMA 8.9. $l\,\Psi\,\Theta = \Theta$.

*Proof.* By induction on $l$. $[\,]\,\Psi\,\Theta = \Theta$.
$[i]\,\Psi\,\Theta = (\widehat{\rho}[\,]).\texttt{vec}\,.\texttt{tk}\,(\Pi[\,]).\texttt{dr}\,(i\cdot\Pi[\,])[\,] = (\widehat{\rho}[\,]).\texttt{vec}\,[\,] = \Theta$.
$([i]\mathbin{+\!\!+}l)\,\Psi\,\Theta = l\,\Psi\,[i]\,\Psi\,\Theta = l\,\Psi\,\Theta = \Theta$ by induction. □

A key property of $\Psi$ is that it truncates forms. Correct indexing by a list from $\mathbf{N}^n$ drops $n$ elements from the form of the array.

PROPOSITION 8.10. $l\,\Psi\,A \neq \Theta \Rightarrow \rho\,(l\,\Psi\,A) = \texttt{dr}\,(\#l)(\rho\,A)$.

*Proof.* By induction on $l$ and $\rho\,A$. When $l = [\,]$

$$\rho\,([\,]\,\Psi\,A) = \rho\,A = (A=\Theta\rightarrow[0];\;\rho\,A) = ([\,]\,\Psi\,A=\Theta\rightarrow[0];\;\texttt{dr}\,(\#[\,])(\rho\,A))$$

When $l = [i]$ and $A = \texttt{scl}\;a$ (therefore $\rho\,A = [\,]$),

$$\begin{array}{rcl}
\rho\,([i]\,\Psi\,(\texttt{scl}\;a)) & = & \rho\,.\,\widehat{\rho}\,(\texttt{dr}\,1[\,]).\texttt{vec}\,.\texttt{tk}\,(\Pi\,(\texttt{dr}\,1[\,])).\texttt{dr}\,(i\cdot\Pi\,(\texttt{dr}\,1[\,]))[a] \\
& = & \rho\,.(\widehat{\rho}[\,]).\texttt{vec}\,.(\texttt{tk}\,1).(\texttt{dr}\,i)[a] \\
& = & \rho\,.(\widehat{\rho}[\,])(i=0\rightarrow\texttt{vec}\,[a];\;\Theta) \\
& = & \rho\,(i=0\rightarrow\texttt{scl}\;a;\;\Theta) \\
& = & (i=0\rightarrow[\,];\;[0]) \\
& = & ([i]\,\Psi\,(\texttt{scl}\;a)\neq\Theta\rightarrow\texttt{dr}\,1[\,];\;[0]) \\
& = & ([i]\,\Psi\,(\texttt{scl}\;a)=\Theta\rightarrow[0];\;(\texttt{dr}\,1).\rho\,.\,\texttt{scl}\;a)
\end{array}$$

When $l = [i]$ and $\rho\,A = [x]\mathbin{+\!\!+}t$

$$\begin{array}{rcl}
\rho\,([i]\,\Psi\,A) & = & \rho\,.(\widehat{\rho}\,t).\texttt{vec}\,.(\texttt{tk}\,(\Pi\,t)).(\texttt{dr}\,(i\cdot\Pi\,t)).\texttt{snd}\;A \\
& = & (\Pi\,t=\#.(\texttt{tk}\,(\Pi\,t)).(\texttt{dr}\,(i\cdot\Pi\,t)).\texttt{snd}\;A\rightarrow t;\;[\#.(\texttt{tk}\,(\Pi\,t)).\texttt{dr}\,(i\cdot\Pi\,t).\texttt{snd}\;A]) \\
& = & (\#.\texttt{dr}\,(i\cdot\Pi\,t).\texttt{snd}\;A\geq\Pi\,t\rightarrow t;\;[\#.(\texttt{tk}\,(\Pi\,t)).(\texttt{dr}\,(i\cdot\Pi\,t)).\texttt{snd}\;A])
\end{array}$$

$$
\begin{aligned}
&= (\,\texttt{siz}\,A \dot{-} (i\cdot\Pi\,t) \geq \Pi\,t \rightarrow t;\; [\#.(\,\texttt{tk}\,(\Pi\,t)).(\,\texttt{dr}\,(i\cdot\Pi\,t)).\,\texttt{snd}\,A])\\
&= ((x\dot{-}i)\cdot\Pi\,t \geq \Pi\,t \rightarrow t;\; [\#.(\,\texttt{tk}\,(\Pi\,t)).(\,\texttt{dr}\,(i\cdot\Pi\,t)).\,\texttt{snd}\,A])\\
&= (i<x \rightarrow \texttt{dr}\,1(\rho\,A);\; [\#.(\,\texttt{tk}\,(\Pi\,t)).(\,\texttt{dr}\,(i\cdot\Pi\,t)).\,\texttt{snd}\,A])\\
&= (i<x \rightarrow \texttt{dr}\,(\#[i])(\rho\,A);\; [\#.(\,\texttt{tk}\,(\Pi\,t)).(\,\texttt{dr}\,(x\cdot\Pi\,t)).\,\texttt{snd}\,A])\\
&= (i<x \rightarrow \texttt{dr}\,(\#[i])(\rho\,A);\; [\#.(\,\texttt{tk}\,(\Pi\,t))[\,]])\\
&= (i<x \rightarrow \texttt{dr}\,(\#[i])(\rho\,A);\; [0])\\
&= ([i]\,\Psi\,A = \Theta \rightarrow [0];\; \texttt{dr}\,(\#[i])(\rho\,A))
\end{aligned}
$$

When the index is of the form $[i] +\!\!+ l$

$$
\begin{aligned}
\rho\,(([i] +\!\!+ l)\,\Psi\,A) &= \rho\,(l\,\Psi\,[i]\,\Psi\,A)\\
&\overset{\text{induc.}}{=} (l\,\Psi\,[i]\,\Psi\,A = \Theta \rightarrow [0];\; (\,\texttt{dr}\,(\#l)).\rho\,([i]\,\Psi\,A))\\
&\overset{\text{induc.}}{=} (l\,\Psi\,[i]\,\Psi\,A = \Theta \rightarrow [0];\; \texttt{dr}\,(\#l)([i]\,\Psi\,A = \Theta \rightarrow [0];\; \texttt{dr}\,1(\rho\,A)))\\
&= (l\,\Psi\,[i]\,\Psi\,A = \Theta \rightarrow [0];\; (\,\texttt{dr}\,(\#l)).(\,\texttt{dr}\,1)(\rho\,A))\\
&= (l\,\Psi\,[i]\,\Psi\,A = \Theta \rightarrow [0];\; \texttt{dr}\,((\,\texttt{siz}\,l)+1)(\rho\,A)) \quad (8.5)\\
&= (([i] +\!\!+ l)\,\Psi\,A = \Theta \rightarrow [0];\; \texttt{dr}\,(\,\texttt{siz}\,([i] +\!\!+ l))(\rho\,A))
\end{aligned}
$$

□

## 9. Mapping and Reduction Operators.

This section defines second-order operators which lift base-type operations to list or array operations. The first pair of operators apply a base-type function to a whole structure of values, the second pair apply a structure of functions to a single value, the third pair reduce a structure of values to a single value and the last pair of operators reduce every prefix of a structure of values.

Definition 15. $()^*\, :\, (S \rightarrow T) \rightarrow S^* \rightarrow T^*$ *maps a base-type operation to a list homomorphism (i.e. pointwise)*

$$g^*[\,] = [\,] \quad \textit{and} \quad g^*([x] +\!\!+ l) = [gx] +\!\!+ (g^*l)$$

Proposition 9.1. $\#(g^*l) = \#l$.

*Proof.* By induction on $l \in S^*$. □

Definition 16. $()^{\Box}\, :\, (S \rightarrow T) \rightarrow S^{\Box} \rightarrow T^{\Box}$ *promotes a base-type operation to a pointwise array operation:*

$$h^{\Box}(f,c) = (f,\, h^*c)$$

From proposition 9.1, it follows that $()^{\Box}$ is well defined: $\#(h^*c) = \#c = \Pi\,f$.

Definition 17. $()_*\, :\, (S \rightarrow T)^* \rightarrow S \rightarrow T^*$ *applies a list of functions to a given argument:*

$$l_*\,s = (\lambda g.\; gs)^*l$$

It follows from proposition 9.1 that $\#(l_*\,s) = \#l$ and so the following operator is well defined.

Definition 18. $()_{\Box}\, :\, (S \rightarrow T)^{\Box} \rightarrow S \rightarrow T^{\Box}$ *applies an array of functions to a given argument:*

$$(f,c)_{\Box}\,s = (f, c_*\,s)$$

Definition 19. $()/_{()} : (S \to T \to S) \to S \to T^* \to S$ *makes a list reduction from a binary operator $g$ and an initial value $e$,*

$$(g/_e)\,[\,] = e, \qquad (g/_e)(l +\!\!+ [t]) = \; g\,(g/_e\, l)\, t.$$

Reduction is usually applied in a less general context to inner operations $(S = T)$ which constitute a monoid. For example $\cdot/_1 \; = \Pi$ and $+/_0 \; = \Sigma$ on $S = T = \mathbf{N}$. The following well-known property is the algebraic basis for parallel sum algorithms (and indirectly parallel prefix algorithms [20, 21]) which require an associative binary operation.

Proposition 9.2. *If $g \in (T \to T \to T)$, $e \in T$ and $(T, g, e)$ is a monoid, then $g/_e$ is a morphism $T^* \to T$ i.e.*

$$g/_e\,(l_1 +\!\!+ l_2) = g\,(g/_e\, l_1)\,(g/_e\, l_2)$$

*Proof.* By induction on $l_2$.

$$\begin{array}{rcl}
(g/_e)(l_1 +\!\!+ [\,]) & = & g/_e\, l_1 \\
 & = & g(g/_e\, l_1)e = g(g/_e\, l_1)(g/_e\,[\,]) \\
g/_e\,(l_1 +\!\!+ (l_2 +\!\!+ [t])) & = & (g/_e)((l_1 +\!\!+ l_2) +\!\!+ [t]) \\
 & = & g((g/_e)(l_1 +\!\!+ l_2))t \\
 & \overset{\text{induct.}}{=} & g(g((g/_e)l_1)((g/_e)l_2))t \\
 & \overset{\text{g assoc.}}{=} & g((g/_e)l_1)(g((g/_e)l_2)t) \\
 & = & g((g/_e)l_1)((g/_e)(l_2 +\!\!+ [t]))
\end{array}$$

□

Definition 20. $()/\!\!\!\square_{()} : (S \to T \to S) \to S \to T^{\square} \to S$ *makes an array reduction from a binary operation $g$ and an initial value $e$.*

$$g/\!\!\!\square_e\,(f, c) = g/_e\, c$$

Array reduction is of interest for its possible parallel implementations. There is a natural algorithm for the reduction of a list of values whose elements are at the nodes of a mesh-connected network: reduce along successive dimensions and then find the result at node $[0, 0, \ldots, 0]$. Since a mesh-connected topology can be encoded as a form $\rho\, A$, it is possible to describe this algorithm for $g/\!\!\!\square_e\, A$ by a set of dimension-recursive equations. See section 13 for more on this topic.

Definition 21. *The prefix operator* `pre` $: T^* \to T^{**}$ *returns the list of non-empty prefixes of a given list.*

$$\texttt{pre}\,[\,] = [\,], \qquad \texttt{pre}\,([t] +\!\!+ l) = [[t]] +\!\!+ (([t] +\!\!+)^*(\texttt{pre}\, l))$$

*For example,* $\texttt{pre}\,[a, b, c] = [[a], [a, b], [a, b, c]]$.

Proposition 9.3. $\#(\texttt{pre}\, l) = \# l$.

*Proof.* By induction on $l$, using proposition 9.1. □

Definition 22. *The list scan operator* $()/\!\!/_{()} : (S \to T \to S) \to S \to T^* \to S^*$ *takes a binary operation $g$, an initial value $e$, and reduces every prefix of a given list:*

$$g/\!\!/_e \; = (g/_e)^*.\texttt{pre}$$

Propositions 9.1 and 9.3 together imply the following.

PROPOSITION 9.4. $\#(g/\!\!/_e\, l) = \#l$.

DEFINITION 23. *The array scan operator has the effect of the list scan for content lists. It takes a binary operation $g$, an initial value $e$, and reduces every prefix of the content list of a given array, $\Box\!\!\!\!/\!\!/ : (S \to T \to S) \to S \to T^{\Box} \to S^{\Box}$.*

$$g\Box\!\!\!\!/\!\!/_e\,(f, c) = (f,\ g/\!\!/_e\, c)$$

By proposition 9.4, $()\Box\!\!\!\!/\!\!/_{()}$ is well defined. Array scan is of interest for its possible parallel implementations, and remarks similar to those following the definition of $()\Box\!\!\!\!/_{()}$ are applicable. Dimension-recursive equations for $g\Box\!\!\!\!/\!\!/_e$ can constitute an algebraic formulation of parallel prefix algorithms for mesh-connected networks. See section 13.

The following are useful formal properties of the map and application operators.

PROPOSITION 9.5. *List mapping distributes over function composition,*

$$g^*.h^* = (g.h)^*$$

*Proof.* By induction on the list argument to either side of the identity.

$$\begin{array}{rcl}
g^*.h^*[\,] & = & g^*[\,] = [\,] = (g.h)^*[\,] \\
g^*.h^*([x] +\!\!+ l) & = & g^*([hx] +\!\!+ (h^*l)) \\
 & = & [g(hx)] +\!\!+ (g^*(h^*l)) \\
 & = & [(g.h)x] +\!\!+ (g^*.h^*l) \\
 & \overset{\text{induc.}}{=} & [(g.h)x] +\!\!+ ((g.h)^*l) \\
 & = & (g.h)^*([x] +\!\!+ l)
\end{array}$$

□

COROLLARY 9.6. *Array mapping distributes over function composition,*

$$g^{\Box}.h^{\Box} = (g.h)^{\Box}$$

*Proof.*

$$g^{\Box}.h^{\Box}(f, c) = (f, g^*.h^*c) = (f, (g.h)^*c) = (g.h)^{\Box}(f, c)$$

□

PROPOSITION 9.7. *List mapping distributes over list application,*

$$g^*.l_* = ((g.)^*l)_*$$

*Proof.*

$$\begin{array}{rcl}
(g^*.l_*)a & = & g^*.(\lambda x.\ xa)^*l \\
 & = & (g.(\lambda x.\ xa))^*l \\
 & = & (\lambda x.\ (g.x)a)^*l \\
 & = & ((\lambda x.\ xa).(g.))^*l \\
 & = & (\lambda x.\ xa)^*((g.)^*l) \\
 & = & ((g.)^*l)_*\, a
\end{array}$$

□

COROLLARY 9.8. *Array mapping distributes over array application,*

$$g^{\square}.A_{\square} = ((g.)^{\square}A)_{\square}$$

*Proof.*

$$\begin{aligned}(g^{\square}.(f,\, c)_{\square})a &= g^{\square}(f,\, c_{*}\, a)\\ &= (f,\, g^{*}.c_{*}\, a)\\ &\overset{9.7}{=} (f,\, ((g.)^{*}c)_{*}\, a)\\ &= (f,\, (g.)^{*}c)_{\square}\, a\\ &= ((g.)^{\square}(f,\, c))_{\square}\, a\end{aligned}$$

□

**10. Outer Product.** The outer product is a second-order operator of type

$$\texttt{out} : (R \to S \to T) \to R^{\square} \to S^{\square} \to T^{\square}.$$

It is a multidimensional generalisation of tensor products. Just as arrays are not matrices, the outer product is not a tensor product, it assumes no vector space structure on its base type. $\texttt{out}\, g$ is also written infix $\otimes_g$, or $\otimes$ when $g$ is known from context or irrelevant. The definition is as follows.

DEFINITION 24.

$$A \otimes_g B = (\rho\, A \mathbin{+\!\!+} \rho\, B, (\mathbin{+\!\!+} /_{[\,]}).((^{*}.g)^{*}(\,\texttt{snd}\, A))_{*}\,(\,\texttt{snd}\, B))$$

For example, let $A = \texttt{vec}\,[a,b]$ and $B = \texttt{vec}\,[1,2]$, then

$$\begin{aligned}\texttt{snd}\,(A \otimes_{+} B) &= (\mathbin{+\!\!+} /_{[\,]}).((^{*}.+)^{*}[a,b])_{*}\,[1,2]\\ &= (\mathbin{+\!\!+} /_{[\,]}).[(a+)^{*}, (b+)^{*}]_{*}\,[1,2]\\ &= (\mathbin{+\!\!+} /_{[\,]})[(a+)^{*}[1,2], (b+)^{*}[1,2]]\\ &= (\mathbin{+\!\!+} /_{[\,]})[[a+1, a+2], [b+1, b+2]]\\ &= [a+1, a+2, b+1, b+2]\end{aligned}$$

and

$$\rho\,(A \otimes_{+} B) = [2] \mathbin{+\!\!+} [2] = [2,2]$$

The caracteristic property of outer products is that when $\#l = \texttt{dim}\, A$ and $\#r = \texttt{dim}\, B$,

$$(l \mathbin{+\!\!+} r)\,\Psi\,(A \otimes B) = (l\,\Psi\, A) \otimes (r\,\Psi\, B)$$

This is a special case of proposition 10.9 and a generalisation of the following property of tensor products (to be precise: of two-dimensional outer products)

$$[i,j,k,l]\,\Psi\,(A \otimes_g B) = \texttt{scl}\,(gxy) \quad \text{where} \quad [i,j]\,\Psi\, A = \texttt{scl}\, x, [k,l]\,\Psi\, B = \texttt{scl}\, y$$

We first prove that the outer product is well defined.

LEMMA 10.1. $\Pi\,.\rho\,(A \otimes_g B) = \#.\,\texttt{snd}\,(A \otimes_g B)$

*Proof.* Let $\Sigma = (+/_0\,)$ in what follows.

$$\begin{aligned}
\#.\,\mathtt{snd}\,(A \otimes_g B) &= \#.(\mathbin{+\!\!+}\,/_{[\,]}\,).((^*.g)^*\,(\,\mathtt{snd}\,A))_*\,(\,\mathtt{snd}\,B) \\
&= \Sigma.\#^*.((^*.g)^*\,(\,\mathtt{snd}\,A))_*\,(\,\mathtt{snd}\,B) \\
&\overset{9.7}{=} \Sigma.((\#.)^*.(^*.g)^*\,(\,\mathtt{snd}\,A))_*\,(\,\mathtt{snd}\,B) \\
&\overset{9.5}{=} \Sigma.((\#.^*.g)_*\,(\,\mathtt{snd}\,A))_*\,(\,\mathtt{snd}\,B) \\
&= \Sigma.((\lambda x.\,\#)^*\,(\,\mathtt{snd}\,A))_*\,(\,\mathtt{snd}\,B) \\
&= \Sigma.((\lambda x.\,\#)^*\,(\iota.\,\mathtt{siz}\,A))_*\,(\,\mathtt{snd}\,B) \\
&= \Sigma((\lambda x.\,\#(\,\mathtt{snd}\,B))^*(\,\mathtt{snd}\,A)) \\
&= (\#(\,\mathtt{snd}\,A)) \cdot (\#(\,\mathtt{snd}\,A)) \\
&= (\Pi\,(\rho\,A)) \cdot (\Pi\,(\rho\,B)) \\
&= \Pi\,((\rho\,A) \mathbin{+\!\!+} (\rho\,B)) \;=\; \Pi\,(\rho\,(A \otimes B))
\end{aligned}$$

$\square$

PROPOSITION 10.2. $\Theta \otimes A = A \otimes \Theta = \Theta$

*Proof.*

$$\begin{aligned}
A \otimes_g \Theta &= ((\rho\,A) \mathbin{+\!\!+} [0],\ (\mathbin{+\!\!+}\,/_{[\,]}\,).((^*.g)^*(\,\mathtt{snd}\,A))_*\,[\,]) \\
&= ([0],\ (\mathbin{+\!\!+}\,/_{[\,]})[\,]) \\
&= ([0],\ [\,]) = \Theta \\
\Theta \otimes_g A &= ([0] \mathbin{+\!\!+} (\rho\,A),\ (\mathbin{+\!\!+}\,/_{[\,]}\,).((^*.g)^*[\,])_*\,(\,\mathtt{snd}\,A)) \\
&= ([0],\ (\mathbin{+\!\!+}\,/_{[\,]}\,).[\,]_*\,(\,\mathtt{snd}\,A)) \\
&= ([0],\ (\mathbin{+\!\!+}\,/_{[\,]})[\,]) = ([0], [\,]) = \Theta
\end{aligned}$$

$\square$

Outer product by a scalar, is equivalent to mapping a base-type product, in accordance with the usual notion of scalar product.

PROPOSITION 10.3. *$(\,\mathtt{scl}\,a) \otimes_g A = (ga)^{\square}A$ and $A \otimes_g (\,\mathtt{scl}\,a) = (g^{\square}A)_{\square}\,a$.*

*Proof.*

$$\begin{aligned}
(\,\mathtt{scl}\,a) \otimes_g A &= ([\,] \mathbin{+\!\!+} (\rho\,A),\ (\mathbin{+\!\!+}\,/_{[\,]}\,).((^*.g)^*([a]))_*\,(\,\mathtt{snd}\,A)) \\
&= (\rho\,A,\ (\mathbin{+\!\!+}\,/_{[\,]}\,)[(ga)^*(\,\mathtt{snd}\,A)]) \\
&= (\rho\,A,\ (ga)^*(\,\mathtt{snd}\,A)) \\
&= (ga)^{\square}A \\
A \otimes_g (\,\mathtt{scl}\,a) &= ((\rho\,A) \mathbin{+\!\!+} [\,],\ (\mathbin{+\!\!+}\,/_{[\,]}\,).((^*.g)^*(\,\mathtt{snd}\,A))_*\,[a]) \\
&= (\rho\,A,\ (\mathbin{+\!\!+}\,/_{[\,]}\,).((\lambda x.\,x[a]).^*.g)^*(\,\mathtt{snd}\,A)) \\
&= (\rho\,A,\ (\mathbin{+\!\!+}\,/_{[\,]}\,).(\lambda x.\,(gx)^*[a])^*(\,\mathtt{snd}\,A)) \\
&= (\rho\,A,\ (\mathbin{+\!\!+}\,/_{[\,]}\,).(\lambda x.\,[gxa])^*(\,\mathtt{snd}\,A)) \\
&= (\rho\,A,\ (\lambda x.\,gxa)^*(\,\mathtt{snd}\,A)) \\
&= (\rho\,A,\ (\lambda x.\,gx)^*(\,\mathtt{snd}\,A))_{\square}\,a \\
&= (\rho\,A,\ g^*(\,\mathtt{snd}\,A))_{\square}\,a \\
&= (g^{\square}A)_{\square}\,a
\end{aligned}$$

$\square$

LEMMA 10.4.

$$\texttt{snd}\,(X \otimes_g Y) = (+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*(\,\texttt{snd}\,Y))^*(\,\texttt{snd}\,X)$$

*Proof.*

$$\begin{aligned}
\texttt{snd}\,(X \otimes_g Y) &= (+\!\!+\,/_{[\,]}).((^*.g)^*(\,\texttt{snd}\,X))_*\,(\,\texttt{snd}\,Y)) \\
&= (+\!\!+\,/_{[\,]}).(\lambda x.\,x(\,\texttt{snd}\,Y))^*.(^*.g)^*(\,\texttt{snd}\,X) \\
&= (+\!\!+\,/_{[\,]}).((\lambda x.\,x(\,\texttt{snd}\,Y)).(^*.g))^*(\,\texttt{snd}\,X) \\
&= (+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*(\,\texttt{snd}\,Y))^*(\,\texttt{snd}\,X)
\end{aligned}$$

□

PROPOSITION 10.5. *If $g : T \to T \to T$ is associative, then $\otimes = \texttt{out}\,g$ is also associative.*

*Proof.* In what follows let $\alpha = \texttt{snd}\,A$, $\beta = \texttt{snd}\,B$ and $\gamma = \texttt{snd}\,C$. Associativity of $g$ can be written as follows

$$g(gab) = (ga).(gb)$$

We will use without proof the following properties of lists of lists

$$h^*.(+\!\!+\,/_{[\,]}) = (+\!\!+\,/_{[\,]}).h^{**}$$

$$(+\!\!+\,/_{[\,]}).(+\!\!+\,/_{[\,]}) = (+\!\!+\,/_{[\,]}).(+\!\!+\,/_{[\,]})^*$$

and use lemma 10.4 to express outer products without $()_*$, which simplifies the proof.

$$\begin{aligned}
\rho\,((A \otimes B) \otimes C) &= (\rho\,A +\!\!+ \rho\,B) +\!\!+ \rho\,C \\
&= \rho\,A +\!\!+ (\rho\,B +\!\!+ \rho\,C) \\
&= \rho\,(A \otimes (B \otimes C)) \\
\texttt{snd}\,((A \otimes B) \otimes C) &= (+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*\gamma)^*(\,\texttt{snd}\,(A \otimes B)) \\
&= (+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*\gamma)^*.(+\!\!+\,/_{[\,]}).(\lambda y.\,(gy)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*\gamma)^{**}.(\lambda y.\,(gy)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*\gamma)^*.(\lambda y.\,(gy)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(+\!\!+\,/_{[\,]})^*.(\lambda y.\,(\lambda x.\,(gx)^*\gamma)^*.(gy)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(+\!\!+\,/_{[\,]}).((\lambda x.\,(gx)^*\gamma).(gy))^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(+\!\!+\,/_{[\,]}).((\lambda x.\,(g(gyx))^*\gamma))^*\beta)^*\alpha \\
&\overset{\text{g assoc.}}{=} (+\!\!+\,/_{[\,]}).(\lambda y.\,(+\!\!+\,/_{[\,]}).((\lambda x.\,(gy)^*.(gx)^*\gamma))^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(+\!\!+\,/_{[\,]}).((gy)^*.(\lambda x.\,(gx)^*\gamma))^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(+\!\!+\,/_{[\,]}).(gy)^{**}.(\lambda x.\,(gx)^*\gamma)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(gy)^*.(+\!\!+\,/_{[\,]}).(\lambda x.\,(gx)^*\gamma)^*\beta)^*\alpha \\
&= (+\!\!+\,/_{[\,]}).(\lambda y.\,(gy)^*.\,\texttt{snd}\,(B \otimes C))^*\alpha \\
&= \texttt{snd}\,(A \otimes (B \otimes C))
\end{aligned}$$

□

COROLLARY 10.6. *If $(T, g, e)$ is a monoid then $(T^{\square}, \otimes_g, \texttt{scl}\ e)$ is also a monoid.*

*Proof.* By propositions 10.5 and 10.3. □

Remark that, even for a commutative $g$, $\otimes_g$ is not commutative.

We now prove the caracteristic property of outer products proposition 10.9. Indexing an outer product by a concatenation is equal to the outer product of the indexings.

First a technical lemma about lists of equal-length lists.

LEMMA 10.7. *Let $T^n = \{A : T^* \mid \#x = n\}$ and $l \in (T^n)^*$ then*

$$(\texttt{tk}\,(a\cdot n)).(\mathbin{+\!\!+}/_{[\,]}) = (\mathbin{+\!\!+}/_{[\,]}).(\texttt{tk}\,a)$$

*and*

$$(\texttt{dr}\,(a\cdot n)).(\mathbin{+\!\!+}/_{[\,]}) = (\mathbin{+\!\!+}/_{[\,]}).(\texttt{dr}\,a)$$

*Proof.* By induction on $l$.

$$\begin{array}{rcl}
(\texttt{tk}\,(a\cdot n)).(\mathbin{+\!\!+}/_{[\,]})[\,] &=& \texttt{tk}\,(a\cdot n)[\,]\\
&=& [\,]\\
&=& (\mathbin{+\!\!+}/_{[\,]})\\
&=& (\mathbin{+\!\!+}/_{[\,]}).(\texttt{tk}\,a)[\,]\\
\texttt{tk}\,(a\cdot n)((\mathbin{+\!\!+}/_{[\,]})([r]\mathbin{+\!\!+}l)) &=& (\texttt{tk}\,(a\cdot n))([r]\mathbin{+\!\!+}(\mathbin{+\!\!+}/_{[\,]})l)\\
&=& [r]\mathbin{+\!\!+}(\texttt{tk}\,(a\cdot n\dot{-}n)).(\mathbin{+\!\!+}/_{[\,]})l\\
&=& [r]\mathbin{+\!\!+}(\texttt{tk}\,((a\dot{-}1)\cdot n)).(\mathbin{+\!\!+}/_{[\,]})l\\
&=& [r]\mathbin{+\!\!+}((\mathbin{+\!\!+}/_{[\,]})l)\\
&=& (\mathbin{+\!\!+}/_{[\,]})([r]\mathbin{+\!\!+}(\texttt{tk}\,(a\dot{-}1)l))\\
&=& (\mathbin{+\!\!+}/_{[\,]})(\texttt{tk}\,a([r]\mathbin{+\!\!+}l))\\
\texttt{dr}\,(a\cdot n)(\mathbin{+\!\!+}/_{[\,]}) &=& \texttt{dr}\,(a\cdot n)[\,]\\
&=& [\,]\\
&=& (\mathbin{+\!\!+}/_{[\,]})[\,]\\
&=& (\mathbin{+\!\!+}/_{[\,]})(\texttt{dr}\,a[\,])\\
\texttt{dr}\,(a\cdot n)(\mathbin{+\!\!+}/_{[\,]}([r]\mathbin{+\!\!+}l)) &=& \texttt{dr}\,(a\cdot n)[r]\mathbin{+\!\!+}(\mathbin{+\!\!+}/_{[\,]})l\\
&=& \texttt{dr}\,(a\cdot n\dot{-}n)((\mathbin{+\!\!+}/_{[\,]})l)\\
&=& \texttt{dr}\,((a\dot{-}1)\cdot n)((\mathbin{+\!\!+}/_{[\,]})l)\\
&=& (\mathbin{+\!\!+}/_{[\,]})(\texttt{dr}\,a([r]\mathbin{+\!\!+}l))
\end{array}$$

□

LEMMA 10.8. *$\#l \le \texttt{dim}\,A \;\Rightarrow\; l\,\Psi\,(A\otimes B) = (l\,\Psi\,A)\otimes B$.*

*Proof.* By induction on l. Let $\alpha = \texttt{snd}\,A$, $\beta = \texttt{snd}\,B$ and $\gamma = \texttt{snd}\,(A\otimes B)$.

1.

$$[\,]\,\Psi\,(A\otimes B) = (A\otimes B) = ([\,]\,\Psi\,A)\otimes B$$

2. $l = [i]$, $\texttt{dim}\,A \ge 1$, using $\gamma = (\mathbin{+\!\!+}/_{[\,]}).(\lambda r.(gr)^*\beta)^*\alpha$
   $[i]\,\Psi\,(A\otimes B) =$

$$
\begin{array}{ll}
= & (\,\widehat{\rho}\,(\,\mathtt{dr}\,1(\rho\,A \mathbin{+\!\!+} \rho\,B))).\,\mathtt{vec}\,.(\,\mathtt{tk}\,(\Pi\ \mathtt{dr}\,1(\rho\,A \mathbin{+\!\!+} \rho\,B))).(\,\mathtt{dr}\,(i\cdot\Pi\ \mathtt{dr}\,1(\rho\,A \mathbin{+\!\!+} \rho\,B)))\gamma \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.(\,\mathtt{tk}\,(\Pi\,(\,\mathtt{dr}\,1\rho\,A)\cdot\Pi\,\rho\,B)).(\,\mathtt{dr}\,(i\cdot\Pi\,(\,\mathtt{dr}\,1\rho\,A)\cdot\Pi\,\rho\,B)\gamma \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.(\,\mathtt{tk}\,(\Pi\,(\,\mathtt{dr}\,1\rho\,A)\cdot\#\beta)).(\,\mathtt{dr}\,(i\cdot\Pi\,(\,\mathtt{dr}\,1\rho\,A)\cdot\#\beta)\gamma \\
\overset{(10.7)}{=} & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.(\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{tk}\,(\Pi\ \mathtt{dr}\,1\rho\,A)).(\,\mathtt{dr}\,(i\cdot\Pi\ \mathtt{dr}\,1\rho\,A)). \\
 & \qquad\quad (\lambda r.(gr)^*\beta)^*\alpha \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.(\mathbin{+\!\!+}/_{[\,]}).(\lambda r.(gr)^*\beta)^*.(\,\mathtt{tk}\,(\Pi\ \mathtt{dr}\,1\rho\,A)). \\
 & \qquad\quad (\,\mathtt{dr}\,(i\cdot\Pi\ \mathtt{dr}\,1\rho\,A))\alpha \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.(\mathbin{+\!\!+}/_{[\,]}).(\lambda r.(gr)^*\beta)^*.\,\mathtt{snd}\ ([i]\,\Psi\,A) \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{vec}\,.\,\mathtt{snd}\,(([i]\,\Psi\,A)\otimes B) \\
= & (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{rav}\,(([i]\,\Psi\,A)\otimes B) \\
= & \widehat{\rho}\,([i]\,\Psi\,A = \Theta \rightarrow (\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} (\rho\,B);\ \rho\,(([i]\,\Psi\,A)\otimes B))(\,\mathtt{rav}\,(([i]\,\Psi\,A)\otimes B)) \\
= & ([i]\,\Psi\,A = \Theta \rightarrow (\,\widehat{\rho}\,((\,\mathtt{dr}\,1\rho\,A) \mathbin{+\!\!+} \rho\,B)).\,\mathtt{rav}\,(([i]\,\Psi\,A)\otimes B) \\
; & \qquad (\,\widehat{\rho}\,(\rho\,(([i]\,\Psi\,A)\otimes B))).\,\mathtt{rav}\,(([i]\,\Psi\,A)\otimes B)) \\
= & ([i]\,\Psi\,A = \Theta \rightarrow \widehat{\rho}\,((\,\mathtt{dr}\,1(\rho\,A)) \mathbin{+\!\!+} (\rho\,B)).\,\mathtt{rav}\,\Theta;\ ([i]\,\Psi\,A)\otimes B) \\
= & ([i]\,\Psi\,A = \Theta \rightarrow \Theta;\ ([i]\,\Psi\,A)\otimes B) \\
= & ([i]\,\Psi\,A = \Theta \rightarrow \Theta\otimes B;\ ([i]\,\Psi\,A)\otimes B) \\
= & ([i]\,\Psi\,A = \Theta \rightarrow ([i]\,\Psi\,A)\otimes B;\ ([i]\,\Psi\,A)\otimes B) \\
= & ([i]\,\Psi\,A)\otimes B
\end{array}
$$

3. $\mathtt{dim}\,A \geq \#l+1,\ ([i]\,\Psi\,A)\otimes B = \Theta$
   $\Rightarrow \rho\,([i]\,\Psi\,A) \mathbin{+\!\!+} \rho\,B = [0] \Rightarrow [i]\,\Psi\,A = \Theta$ or $B = \Theta$.

$$
\begin{array}{rcll}
([i] \mathbin{+\!\!+} l)\,\Psi\,(A\otimes B) & = & l\,\Psi\,[i]\,\Psi\,(A\otimes B) & \\
 & = & l\,\Psi\,(([i]\,\Psi\,A)\otimes B) & \text{induction} \\
 & = & ([i]\,\Psi\,A = \Theta \rightarrow \Theta;\ (B = \Theta \rightarrow \Theta;\ \Theta)) & \\
 & = & ([i]\,\Psi\,A = \Theta \rightarrow (l\,\Psi\,\Theta)\otimes B;\ (B = \Theta \rightarrow (l\,\Psi\,[i]\,\Psi\,A)\otimes B;\ \Theta)) & \\
 & = & ([i]\,\Psi\,A = \Theta \rightarrow (l\,\Psi\,[i]\,\Psi\,A)\otimes B;\ (B = \Theta \rightarrow (l\,\Psi\,[i]\,\Psi\,A)\otimes B;\ \Theta)) & \\
 & = & (l\,\Psi\,[i]\,\Psi\,A)\otimes B & \\
 & = & (([i] \mathbin{+\!\!+} l)\,\Psi\,A)\otimes B &
\end{array}
$$

4. $\mathtt{dim}\,A \geq \#l+1,\ ([i]\,\Psi\,A)\otimes B \neq \Theta$
   $\Rightarrow [i]\,\Psi\,A \neq \Theta$ and $B \neq \Theta$
   $\Rightarrow \mathtt{dim}\,(([i]\,\Psi\,A)\otimes B) = \mathtt{dim}\,A \dot{-} 1 + \mathtt{dim}\,B \geq (\#l+1)\dot{-}1 + \mathtt{dim}\,B \geq \#l.$

$$
\begin{array}{rcll}
([i] \mathbin{+\!\!+} l)\,\Psi\,(A\otimes B) & = & l\,\Psi\,[i]\,\Psi\,(A\otimes B) & \\
 & = & l\,\Psi\,(([i]\,\Psi\,A)\otimes B) & \text{induction} \\
 & = & (l\,\Psi\,[i]\,\Psi\,A)\otimes B) & \text{induction and } \mathtt{dim}\,(([i]\,\Psi\,A)\otimes B) \geq \#l \\
 & = & (([i] \mathbin{+\!\!+} l)\,\Psi\,A)\otimes B &
\end{array}
$$

□

Proposition 10.9. $l\,\Psi\,(A\otimes B) = ((\,\mathtt{tk}\,(\mathtt{dim}\,A)l)\,\Psi\,A)\otimes((\,\mathtt{dr}\,(\mathtt{dim}\,A)l)\,\Psi\,B)$

*Proof.* $l\,\Psi\,(A\otimes B)$

$$
\begin{array}{ll}
= & ((\,\mathtt{tk}\,(\mathtt{dim}\,A)l) \mathbin{+\!\!+} (\,\mathtt{dr}\,(\mathtt{dim}\,A)l)\,\Psi\,(A\otimes B) \\
= & (\,\mathtt{dr}\,(\mathtt{dim}\,A)l)\,\Psi\,(\,\mathtt{tk}\,(\mathtt{dim}\,A)l)\,\Psi\,(A\otimes B) \\
= & (\,\mathtt{dr}\,(\mathtt{dim}\,A)l)\,\Psi\,((\,\mathtt{tk}\,(\mathtt{dim}\,A)l)\,\Psi\,A)\otimes B) \quad (10.8) \\
= & (\,\mathtt{dr}\,(\mathtt{dim}\,A)l)\,\Psi\,(\#l \geq \mathtt{dim}\,A \rightarrow (\,\mathtt{tk}\,(\mathtt{dim}\,A)l)\,\Psi\,A)\otimes B);\ (l\,\Psi\,A)\otimes B)
\end{array}
$$

$$
\begin{aligned}
&= (\#l \geq \texttt{dim}\,A \rightarrow (\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A) \otimes B);\;\Theta\,\Psi\,((l\,\Psi\,A) \otimes B))\\
&= (\#l \geq \texttt{dim}\,A \rightarrow (\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A) \otimes B);\;(l\,\Psi\,A) \otimes B)\\
&= (\#\,\texttt{tk}\,(\texttt{dim}\,A)l = \texttt{dim}\,A \rightarrow ((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A = \;\texttt{scl}\;(a) \rightarrow X;\; I);\;\Theta)\\
&\quad \text{where} \quad I = ((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A = \Theta \rightarrow Y;\; Z)\\
Z &= (l\,\Psi\,A) \otimes B\\
&= ((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A) \otimes ((\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B)\\
Y &= (\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,(\Theta \otimes B)\\
&= \Theta\\
&= \Theta \otimes ((\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B)\\
&= ((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A) \otimes ((\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B)\\
X &= (\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,((\,\texttt{scl}\;a) \otimes B)\\
&= (\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,(ga)^{\square}B\\
&= (ga)^{\square}(\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B\\
&= (\,\texttt{scl}\;a) \otimes ((\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B)\\
&= ((\,\texttt{tk}\,(\texttt{dim}\,A)l)\,\Psi\,A) \otimes ((\,\texttt{dr}\,(\texttt{dim}\,A)l)\,\Psi\,B)
\end{aligned}
$$

□

**11. Generalised Sum.** We now define a second-order operator which lifts a base-type binary operation to a pointwise binary operation on arrays. It generalises matrix addition, and will be called the generalised sum, an operator of type

$$\texttt{sum} : (R \rightarrow S \rightarrow T) \rightarrow R^{\square} \rightarrow S^{\square} \rightarrow T^{\square}$$

$\texttt{sum}\,gAB$ may also be written $A \oplus_g B$ or $A \oplus B$ if $g$ is known from context or is irrelevant. Matrix addition is undefined for matrices of unequal forms, and $\oplus$ behaves similarly. It returns $\Theta$ if forms don't match. The effect of $\oplus_g$ on content lists is to construct pairs of corresponding elements and then apply to each the uncurried version of $g$. This will require the following two definitions.

Definition 25.

$$
\begin{aligned}
\texttt{zip} \quad &: \quad R^* \rightarrow S^* \rightarrow (R \times S)^*\\
\texttt{zip}\,[\,]\,l \quad &= \quad [\,]\\
\texttt{zip}\,l\,[\,] \quad &= \quad [\,]\\
\texttt{zip}\,([x_1] +\!\!+ l_1)\,([x_2] +\!\!+ l_2) \quad &= \quad [(x_1, x_2)] +\!\!+ (\,\texttt{zip}\,l_1\,l_2)
\end{aligned}
$$

Lemma 11.1. $\#(\,\texttt{zip}\,l_1\,l_2) = (\#l_1)\min(\#l_2)$

*Proof.* By induction on $l_1$. □

The $\texttt{unc}$ operator is required to map curried functions onto lists of pairs returned by $\texttt{zip}$.

Definition 26.

$$
\begin{aligned}
\texttt{unc} \quad &: \quad (R \rightarrow S \rightarrow T) \rightarrow (R \times S) \rightarrow T\\
\texttt{unc}\,g\,(r, s) \quad &= \quad grs
\end{aligned}
$$

As we only use functions $g : R \to S \to T$ which are total on $R$ and return total functions $gr$ on $S$, $\texttt{unc}\, g$ will always be total on $R \times S$.

The following elementary properties of $\texttt{zip}$ are analogous to

$$h^*.(+\!\!+ /_{[\,]}) = (+\!\!+ /_{[\,]}).h^{**}$$

on lists of lists, and will be needed in the proof of the distributive property for the outer product.

LEMMA 11.2. *If* $l_1, l_2 \in T^{**}$, $x_1, x_2 \in T^*$, $\#l_1 = \#l_2$ *and* $\#x_1 = \#x_2$ *then*

$$(\texttt{zip}\, x_1\, x_2) +\!\!+ (\texttt{zip}\, l_1\, l_2) = \texttt{zip}\, ([x_1] +\!\!+ l_1)\, ([x_2] +\!\!+ l_2)$$

*Proof.* By induction on $\#x_i$.

$$\begin{array}{rcl}
(\texttt{zip}\, [\,][\,]) +\!\!+ (\texttt{zip}\, l_1\, l_2) & = & [\,] +\!\!+ (\texttt{zip}\, l_1\, l_2) \\
 & = & \texttt{zip}\, l_1\, l_2 \;=\; \texttt{zip}\, ([\,] +\!\!+ l_1)([\,] +\!\!+ l_2) \\
(\texttt{zip}\, ([a_1] +\!\!+ x_1)([a_2] +\!\!+ x_2)) +\!\!+ (\texttt{zip}\, l_1\, l_2) & = & [(a_1, a_2)] +\!\!+ (\texttt{zip}\, x_1\, x_2) +\!\!+ (\texttt{zip}\, l_1\, l_2) \\
 & \overset{\text{induc.}}{=} & [(a_1, a_2)] +\!\!+ (\texttt{zip}\, (x_1 +\!\!+ l_1)(x_2 +\!\!+ l_2)) \\
 & = & \texttt{zip}\, ([a_1] +\!\!+ x_1 +\!\!+ l_1)([a_2] +\!\!+ x_2 +\!\!+ l_2)
\end{array}$$

□

LEMMA 11.3. *Let* $l_1, l_2 \in T^{**}$ *and* $\#l_1 = \#l_2$ *then*

$$(+\!\!+ /_{[\,]}).(\texttt{unc zip})^*(\texttt{zip}\, l_1\, l_2) = \texttt{zip}\, ((+\!\!+ /_{[\,]})\, l_1)\, ((+\!\!+ /_{[\,]})\, l_2)$$

*Proof.* By induction on $\#l_i$.

$$\begin{array}{rcl}
(+\!\!+ /_{[\,]}).(\texttt{unc zip})^*(\texttt{zip}\, [\,]\, [\,]) & = & (+\!\!+ /_{[\,]}).(\texttt{unc zip})^*[\,] \\
 & = & [\,] \;=\; \texttt{zip}\, [\,]\, [\,] \\
 & = & \texttt{zip}\, ((+\!\!+ /_{[\,]})[\,])\, ((+\!\!+ /_{[\,]})[\,]) \\
(+\!\!+ /_{[\,]}).(\texttt{unc zip})^*(\texttt{zip}\, ([x_1] +\!\!+ l_1)\, ([x_2] +\!\!+ l_2)) & = & (+\!\!+ /_{[\,]}).(\texttt{unc zip})^*[(x_1, x_2)] +\!\!+ (\texttt{zip}\, l_1\, l_2) \\
 & = & (+\!\!+ /_{[\,]})\; [\texttt{zip}\, x_1\, x_2] +\!\!+ (\texttt{unc}\, g)^*(\texttt{zip}\, l_1\, l_2) \\
 & = & (\texttt{zip}\, x_1\, x_2) +\!\!+ (+\!\!+ /_{[\,]}).(\texttt{unc}\, g)^*(\texttt{zip}\, l_1\, l_2) \\
 & \overset{\text{induc.}}{=} & (\texttt{zip}\, x_1\, x_2) +\!\!+ \texttt{zip}\, ((+\!\!+ /_{[\,]})l_1)\, ((+\!\!+ /_{[\,]})l_2) \\
 & \overset{11.2}{=} & \texttt{zip}\, ([x_1] +\!\!+ ((+\!\!+ /_{[\,]})l_1))\, ([x_2] +\!\!+ ((+\!\!+ /_{[\,]})l_2)) \\
 & = & \texttt{zip}\, ((+\!\!+ /_{[\,]})([x_1] +\!\!+ l_1))\, ((+\!\!+ /_{[\,]})([x_2] +\!\!+ l_2))
\end{array}$$

□

Now the definition of $\texttt{sum}$.

DEFINITION 27.

$$\begin{array}{rcl}
\texttt{sum} & : & (R \to S \to T) \to R^{\square} \to S^{\square} \to T^{\square} \\
\texttt{sum}\, gAB & = & (\rho\, A = \rho\, B \to (\rho\, A, (\texttt{unc}\, g)^*(\texttt{zip}\, (\texttt{snd}\, A)(\texttt{snd}\, B)));\; \Theta)
\end{array}$$

Lemma 11.4. $\mathtt{sum}$ *is well defined:*

$$\forall g \in (R \to S \to T).\ \forall A \in R^{\square}.\ \forall B \in S^{\square}.\ \mathtt{sum}\, gAB = (A \oplus_g B) \in T^{\square}$$

*Proof.* If $A$ and $B$ have different forms then their sum is $\Theta \in T^{\square}$. Otherwise let $f = \rho\, A = \rho\, B = \rho\,(A \oplus_g B)$ so that $\mathtt{siz}\, X = \#(\,\mathtt{snd}\, X) = \Pi\, f$ for $X = A, B$.

$$\begin{aligned}
\#.\,\mathtt{snd}\,(A \oplus_g B) &= \#.(\,\mathtt{unc}\ g)^*(\,\mathtt{zip}\ (\,\mathtt{snd}\ A)(\,\mathtt{snd}\ B)) \\
&= \#(\,\mathtt{zip}\ (\,\mathtt{snd}\ A)(\,\mathtt{snd}\ B)) \\
&\overset{11.1}{=} (\#(\,\mathtt{snd}\ A))\min(\#(\,\mathtt{snd}\ B)) \\
&= (\Pi\, f)\min(\Pi\, f) \quad = \Pi\, f \\
&= \Pi\,(\rho\,(A \oplus_g B))
\end{aligned}$$

□

The empty array is not a neutral element for $\oplus$ as a zero matrix would be. It is rather absorbent for $\oplus$.

Proposition 11.5. $A \oplus_g \Theta = \Theta \oplus_g A = \Theta$

*Proof.*

$$\begin{aligned}
A \oplus_g \Theta &= (\rho\, A = [0] \to ([0], (\,\mathtt{unc}\ g)^*(\,\mathtt{zip}\ (\,\mathtt{snd}\ A)[\,]));\ \Theta) \\
&= (\rho\, A = [0] \to ([0], (\,\mathtt{unc}\ g)^*[\,]);\ \Theta) \\
&= (\rho\, A = [0] \to ([0], [\,]);\ \Theta) \\
&= (\rho\, A = [0] \to \Theta;\ \Theta) = \quad \Theta \\
\Theta \oplus_g A &= ([0] = \rho\, A \to ([0], (\,\mathtt{unc}\ g)^*(\,\mathtt{zip}\ [\,](\,\mathtt{snd}\ A)));\ \Theta) \\
&= ([0] = \rho\, A \to ([0], (\,\mathtt{unc}\ g)^*[\,]);\ \Theta) \\
&= ([0] = \rho\, A \to ([0], [\,]);\ \Theta) \\
&= ([0] = \rho\, A \to \Theta;\ \Theta) = \quad \Theta
\end{aligned}$$

□

This last property shared by $\otimes$ and $\oplus$ illustrates the fact that $\oplus$ is an abstract product, despite its name. We now consider algebraic properties inherited from $g$ by $\oplus_g$.

If $(T, g)$ has a neutral element $e$, the only possible neutral element for $(T^{\square}, \oplus_g)$ is an array $E$ whose entries are all equal to $e$; but whenever $\rho\, A \neq \rho\, E$ and $A \neq \Theta$, the sum of $A$ and $E$ will not be $A$. There is therefore no neutral element for $\oplus_g$.

Proposition 11.6. *If* $g : T \to T \to T$ *is associative, then* $\oplus = \mathtt{sum}\, g$ *is also associative.*

*Proof.* If $\rho\, A, \rho\, B, \rho\, C$ are not all equal then $A \oplus (B \oplus C) = \Theta = (A \oplus B) \oplus C$. Assume therefore

$$f = \rho\, A = \rho\, B = \rho\, C = \rho\,(A \oplus (B \oplus C)) = \rho\,((A \oplus B) \oplus C)$$

Let also $\alpha = \ \mathtt{snd}\ A$, $\beta = \ \mathtt{snd}\ B$ and $\gamma = \ \mathtt{snd}\ C$.

$$\begin{aligned}
\mathtt{snd}\ (A \oplus (B \oplus C)) &= (\,\mathtt{unc}\ g)^*(\,\mathtt{zip}\ \alpha((\,\mathtt{unc}\ g)^*(\,\mathtt{zip}\ \beta\,\gamma))) \\
&= (\,\mathtt{unc}\ g)^*.(\,\mathtt{zip}\ \alpha).((\,\mathtt{unc}\ g)^*)\ (\,\mathtt{zip}\ \beta\,\gamma) \\
&= (\,\mathtt{unc}\ g)^*.(\lambda(x,(y,z)).\,(x, gyz))^*.(\,\mathtt{zip}\ \alpha)\ (\,\mathtt{zip}\ \beta\,\gamma) \\
&\overset{9.5}{=} (\lambda(x,(y,z)).\,gx(gyz))^*.(\,\mathtt{zip}\ \alpha)\ (\,\mathtt{zip}\ \beta\,\gamma)
\end{aligned}$$

$$\begin{aligned}
&\overset{g\text{ assoc.}}{=} && (\lambda(x,(y,z)).\, g(gxy)z))^*.(\,\mathtt{zip}\;\alpha)\;(\,\mathtt{zip}\;\beta\,\gamma) \\
&= && (\lambda(x,(y,z)).\,(\,\mathtt{unc}\;g)((\,\mathtt{unc}\;g)(x,y),z))^*.(\,\mathtt{zip}\;\alpha)\;(\,\mathtt{zip}\;\beta\,\gamma) \\
&= && (\,\mathtt{unc}\;g)^*.(\lambda(x,(y,z)).\,((\,\mathtt{unc}\;g)(x,y),z))^*.(\,\mathtt{zip}\;\alpha)\;(\,\mathtt{zip}\;\beta\,\gamma) \\
&= && (\,\mathtt{unc}\;g)^*.(\lambda((x,y),z).\,((\,\mathtt{unc}\;g)(x,y),z))^*.(\lambda(x,(y,z).\,((x,y),z))^* \\
& && \qquad\qquad (\,\mathtt{zip}\;\alpha\,(\,\mathtt{zip}\;\beta\,\gamma)) \\
&= && (\,\mathtt{unc}\;g)^*.(\lambda((x,y),z).\,((\,\mathtt{unc}\;g)(x,y),z))^*.(\,\mathtt{zip}\;(\,\mathtt{zip}\;\alpha\beta)\gamma) \\
&= && (\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;((\lambda(x,y),\;(\,\mathtt{unc}\;g)(x,y))^*(\,\mathtt{zip}\;\alpha\,\beta)\gamma) \\
&= && (\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;((\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;\alpha\beta))\gamma) \\
&= && (\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;(\,\mathtt{snd}\;(A\oplus B))\gamma) \\
&= && \mathtt{snd}\;((A\oplus B)\oplus C)
\end{aligned}$$

□

PROPOSITION 11.7. *If $g : T \to T \to T$ is commutative, then $\oplus = \mathtt{sum}\,g$ is also commutative.*

*Proof.* If $\rho\,A \neq \rho\,B$ then $A \oplus B = \Theta = B \oplus A$. Assume therefore

$$f = \rho\,A = \rho\,B = \rho\,(A \oplus B) = \rho\,(B \oplus A)$$

Let also $\alpha = \mathtt{snd}\;A$ and $\beta = \mathtt{snd}\;B$.

$$\begin{aligned}
\mathtt{snd}\;(A\oplus B) &= && (\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;\alpha\,\beta) \\
&= && (\lambda(x,y).\,gxy)^*(\,\mathtt{zip}\;\alpha\,\beta) \\
&\overset{g\text{ comm.}}{=} && (\lambda(x,y).\,gyx)^*(\,\mathtt{zip}\;\alpha\,\beta) \\
&= && (\lambda(x,y).\,gxy)^*.(\lambda(x,y).\,(y,x))^*(\,\mathtt{zip}\;\alpha\,\beta) \\
&= && (\lambda(x,y).\,gxy)^*(\,\mathtt{zip}\;\beta\,\alpha) \\
&= && (\,\mathtt{unc}\;g)^*(\,\mathtt{zip}\;\beta\,\alpha) \\
&= && \mathtt{snd}\;(B\oplus A)
\end{aligned}$$

□

COROLLARY 11.8. *If $(T,g)$ is a commutative semigroup then $(T^\square,\oplus_g)$ is also a commutative semigroup.*

PROPOSITION 11.9. *If $g : T \to T \to T$ is left-distributive over $h : T \to T \to T$ then $\otimes_g$ is left-distributive over $\oplus_h$.*

*Proof.* Assume $g$ distributes over $h$: $gx(hyz) = h(gxy)(gxz)$, and let $A, B, C \in T^\square$ and $\alpha = \mathtt{snd}\;A$, $\beta = \mathtt{snd}\;B$ and $\gamma = \mathtt{snd}\;C$. If $\rho\,B \neq \rho\,C$ then $A \otimes_g (B \oplus_h C) = A \otimes_g \Theta = \Theta$ and the shapes of $A \otimes_g B$ and $A \otimes_g C$ are also unequal, which implies $(A \otimes_g B) \oplus_h (A \otimes_g C) = \Theta$. Distributivity then holds.

The other possibility is that $f = \rho\,B = \rho\,C = \rho\,(B \oplus C)$ and then

$$\rho\,(A \otimes (B \oplus C)) = (\rho\,A) +\!\!+ f = \rho\,(A \otimes B) = \rho\,(A \otimes C) = \rho\,((A \otimes B) \oplus (A \otimes C))$$

so forms are then equal. We now show that content are also equal.

$$\begin{aligned}
\mathtt{snd}\;(A\otimes_g(B\oplus_h C)) &= && (+\!\!+\,/_{[\,]}).((^*.g)^*\alpha)_*.(\,\mathtt{unc}\;h)^*(\,\mathtt{zip}\;\beta\,\gamma) \\
&= && (+\!\!+\,/_{[\,]}).(\lambda x.\,x.(\,\mathtt{unc}\;h)^*(\,\mathtt{zip}\;\beta\,\gamma))^*\;((^*.g)^*\alpha) \\
&= && (+\!\!+\,/_{[\,]}).(\lambda x.\;(gx)^*.(\,\mathtt{unc}\;h)^*(\,\mathtt{zip}\;\beta\,\gamma))^*\alpha
\end{aligned}$$

$$\begin{aligned}
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\ (\lambda(y,z).\,gx(hyz))^*(\,\mathtt{zip}\,\beta\,\gamma))^*\alpha\\
&\overset{\text{distrib.}}{=} (\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\ (\lambda(y,z).\,h(gxy)(gxz))^*(\,\mathtt{zip}\,\beta\,\gamma))^*\alpha\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\ (\,\mathtt{unc}\ h)^*.(\lambda(y,z).\,(gxy,gxz))^*(\,\mathtt{zip}\,\beta\,\gamma))^*\alpha\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{unc}\ h)^{**}.(\lambda x.\ \mathtt{zip}\ ((gx)^*\beta)((gx)^*\gamma))^*\alpha\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\ \mathtt{zip}\ (x\beta)(x\gamma))^*.(^*.g)^*\alpha\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\ \mathtt{zip}\ (x\beta)(x\gamma))^*\ ((^*.g)^*\alpha)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).((\,\mathtt{unc}\ \mathtt{zip}\,).(\lambda x.\ (x\beta,x\gamma)))^*\ ((^*.g)^*\alpha)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{unc}\ \mathtt{zip}\,)^*.(\lambda x.\ (x\beta,x\gamma))^*\ ((^*.g)^*\alpha)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{unc}\ \mathtt{zip}\,)^*\ (\mathtt{zip}\ ((\lambda x.\,x\beta)^*\alpha')((\lambda x.\,x\gamma)^*\alpha'))\\
&\qquad\quad \text{where } \alpha' = (^*.g)^*\alpha\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{unc}\ \mathtt{zip}\,)^*\ (\mathtt{zip}\ (\alpha'_*\,\beta)(\alpha'_*\,\gamma))\\
&\overset{11.3}{=} (\,\mathtt{unc}\ h)^*(\,\mathtt{zip}\ ((\mathbin{+\!\!+}/_{[\,]}).((^*.g)^*\alpha)_*\,\beta)\ ((\mathbin{+\!\!+}/_{[\,]}).((^*.g)^*\alpha)_*\,\gamma))\\
&= (\,\mathtt{unc}\ h)^*(\,\mathtt{zip}\ (\,\mathtt{snd}\ (A\otimes B))(\,\mathtt{snd}\ (A\otimes C)))\\
&= \mathtt{snd}\ ((A\otimes_g B)\oplus_h (A\otimes_g C))
\end{aligned}$$

□

Proposition 11.10. *If $g : T \to T \to T$ is right-distributive over $h : T \to T \to T$ then $\otimes_g$ is right-distributive over $\oplus_h$.*

*Proof.* Assume $g$ distributes over $h$: $g(hxy)z = h(gxz)(gyz)$, and let $A, B, C \in T^{\Box}$ and $\alpha = \mathtt{snd}\,A$, $\beta = \mathtt{snd}\,B$ and $\gamma = \mathtt{snd}\,C$. If $\rho\,A \neq \rho\,B$ then $((A\oplus B)\otimes C) = \Theta \otimes C = \Theta$ and the shapes of $A\otimes C$ and $B \otimes C$ are also unequal, which implies $(A\otimes C)\oplus(B\otimes C) = \Theta$. Distributivity then holds.

The other possibility is that $f = \rho\,A = \rho\,B = \rho\,(A\oplus B)$ and then

$$\rho\,((A\oplus B)\otimes C)) = f \mathbin{+\!\!+} (\rho\,C) = \rho\,(A\otimes C) = \rho\,(B\otimes C) = \rho\,((A\otimes C)\oplus(B\otimes C))$$

so forms are then equal. We now show that contents are also equal.

$$\begin{aligned}
\mathtt{snd}\ ((A\oplus_h B)\otimes_g C) &= (\mathbin{+\!\!+}/_{[\,]}).((^*.g)^*.\,\mathtt{snd}\ (A\oplus_h B))_*\,\gamma\\
&= (\mathbin{+\!\!+}/_{[\,]}).((^*.g)^*.(\,\mathtt{unc}\ h)^*(\,\mathtt{zip}\,\alpha\,\beta))_*\,\gamma\\
&= (\mathbin{+\!\!+}/_{[\,]}).((^*.g.(\,\mathtt{unc}\ h))^*(\,\mathtt{zip}\,\alpha\,\beta))_*\,\gamma\\
&= (\mathbin{+\!\!+}/_{[\,]}).((\lambda(x,y).\,(g(hxy))^*)^*(\,\mathtt{zip}\,\alpha\,\beta))_*\,\gamma\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda x.\,x\gamma)^*((\lambda(x,y).\,(g(hxy))^*)^*(\,\mathtt{zip}\,\alpha\,\beta))\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(g(hxy))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\lambda z.\,g(hxy)z)^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&\overset{\text{distr.}}{=} (\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\lambda z.\,h(gxz)(gyz))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\lambda z.\,(\,\mathtt{unc}\ h)(gxz,gyz))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\,\mathtt{unc}\ h)^*.(\lambda z.\,(gxz,gyz))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\mathbin{+\!\!+}/_{[\,]}).(\,\mathtt{unc}\ h)^{**}.(\lambda(x,y).\,(\lambda z.\,(gxz,gyz))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\lambda z.\,(gxz,gyz))^*\gamma)^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\ \mathtt{zip}\ ((gx)^*\gamma)\ ((gy)^*\gamma))^*\ (\,\mathtt{zip}\,\alpha\,\beta)\\
&= (\,\mathtt{unc}\ h)^*.(\mathbin{+\!\!+}/_{[\,]}).(\lambda(x,y).\,(\,\mathtt{unc}\ \mathtt{zip}\,)((gx)^*\gamma,(gy)^*\gamma))^*\ (\,\mathtt{zip}\,\alpha\,\beta)
\end{aligned}$$

$$\begin{array}{cl}
= & (\,\texttt{unc}\;h)^*.(+\!\!+/_{[\,]}).(\,\texttt{unc zip}\,)^*.(\lambda(x,y).\,((gx)^*\gamma,(gy)^*\gamma))^*\;(\,\texttt{zip}\;\alpha\,\beta) \\
= & (\,\texttt{unc}\;h)^*.(+\!\!+/_{[\,]}).(\,\texttt{unc zip}\,)^*\;(\,\texttt{zip}\;((\lambda x.\,(gx)^*\gamma)^*\alpha)((\lambda x.\,(gx)^*\gamma)^*\alpha)) \\
= & (\,\texttt{unc}\;h)^*.(+\!\!+/_{[\,]}).(\,\texttt{unc zip}\,)^*\;(\,\texttt{zip}\;(((^*.g)^*\alpha)_*\,\gamma)(((^*.g)^*\beta)_*\,\gamma)) \\
\overset{11.3}{=} & (\,\texttt{unc}\;h)^*\;(\,\texttt{zip}\;((+\!\!+/_{[\,]}).((^*.g)^*\alpha)_*\,\gamma)((+\!\!+/_{[\,]}).((^*.g)^*\beta)_*\,\gamma)) \\
= & (\,\texttt{unc}\;h)^*\;(\,\texttt{zip}\,(\,\texttt{snd}\;(A\otimes C))\,(\,\texttt{snd}\;(B\otimes C))) \\
= & \texttt{snd}\;((A\otimes C)\oplus(B\otimes C))
\end{array}$$

□

**12. Stacking Arrays.** We define a binary *stack* operation on arrays which concatenates contents but, because of its effect on forms, is not associative. It plays a critical role in allowing recursive definitions for array operations.

DEFINITION 28.

$$\begin{array}{rcl}
\texttt{stk} & : & T^{\square}\to T^{\square}\to T^{\square} \\
A\,\texttt{stk}\,B & = & (\,\widehat{\rho}\,(s(\rho\,A)(\rho\,B))).\,\texttt{vec}\;((\,\texttt{snd}\;A)+\!\!+(\,\texttt{snd}\;B))
\end{array}$$

*where* $s:\mathcal{F}\to\mathcal{F}\to\mathcal{F}$ *is defined by*

$$\begin{array}{rcl}
s\,[0]\,y & = & y \\
s\,x\,[0] & = & x
\end{array}$$

*else if* $i\neq 0, x\neq[0]$ *then*

$$\begin{array}{rcl}
s\,x\,x & = & [2]+\!\!+x \\
s\,x\,([i]+\!\!+x) & = & [i+1]+\!\!+x \\
s\,([i]+\!\!+x)\,x & = & [i+1]+\!\!+x
\end{array}$$

*else*

$$s\,x\,y = [0].$$

Examples of `stk`:

$$\begin{array}{rcl}
(\,\texttt{scl}\;a)\,\texttt{stk}\,(\,\texttt{scl}\;b) & = & \texttt{vec}\;[a,b] \\
(\,\texttt{scl}\;a)\,\texttt{stk}\,(\,\texttt{vec}\;[b,c]) & = & \texttt{vec}\;[a,b,c] \\
(\,\texttt{vec}\;[a,b])\,\texttt{stk}\,(\,\texttt{vec}\;[c,d]) & = & ([2,2],[a,b,c,d])
\end{array}$$

and

$$(((\,\texttt{vec}\;[a,b])\,\texttt{stk}\,(\,\texttt{vec}\;[c,d]))\,\texttt{stk}\,(\,\texttt{vec}\;[e,f]))\,\texttt{stk}\,(\,\texttt{vec}\;[g,h])=([4,2],[a,b,c,d,e,f,g,h])$$

but

$$((\,\texttt{vec}\;[a,b])\,\texttt{stk}\,(\,\texttt{vec}\;[c,d]))\,\texttt{stk}\,((\,\texttt{vec}\;[e,f])\,\texttt{stk}\,(\,\texttt{vec}\;[g,h]))=([2,2,2],[a,b,c,d,e,f,g,h])$$

PROPOSITION 12.1. $\Theta\,\texttt{stk}\,A = A\,\texttt{stk}\,\Theta = A$

Stacking reconstructs an array from its slices of one fewer dimension. For example if $\texttt{tk}\,1(\rho\,A)=[3]$ then $(\,\texttt{stk}/_{\Theta}\,)[[0]\,\Psi\,A,\;[1]\,\Psi\,A,\;[2]\,\Psi\,A]=A$.

Proposition 12.2 (Stacking). *If* $\mathtt{dim}\, A > 0$ *then*

$$(\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*.(+\!\!+\,/_{[\,]}\,).\iota^*.(\,\mathtt{tk}\,1).\rho\, A)_*\, A = A$$

Lemma 12.3. *If* $\rho\, A = [n] +\!\!+ t$, $x \in \mathbf{N}$ *and* $x \le n$ *then*

$$(\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*.\iota\; x)_*\, A = (\,\widehat{\rho}\,([x] +\!\!+ t)).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(x \cdot \Pi\, t)).\,\mathtt{snd}\; A$$

*Proof.* By induction on $x$.
If $x = 0$ then

$$\begin{aligned}(\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*.\iota 0)_*\, A &= (\,\mathtt{stk}\,/_\Theta\,).[\,]_*\, A\\ &= (\,\mathtt{stk}\,/_\Theta\,)[\,]\\ &= \Theta\\ &= \widehat{\rho}\,([0] +\!\!+ t)X\end{aligned}$$

where $X$ is any array, and in particular this equals the right expression of the identity to prove. Assume now the identity true for $x$ and consider it for $x+1$:

$$\begin{aligned} & (\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*((\iota x) +\!\!+ [x]))_*\, A = \\ = \quad & (\,\mathtt{stk}\,/_\Theta\,).((\,\Psi\,^*.[\_]^*.\iota\; x) +\!\!+ [[x]\,\Psi\,])_*\, A\\ = \quad & ((\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*.\iota\; x)_*\, A)\;\mathtt{stk}\,([x]\,\Psi\, A)\\ \overset{\text{induc.}}{=} \quad & ((\,\widehat{\rho}\,([x] +\!\!+ t)).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(x \cdot \Pi\, t)).\,\mathtt{snd}\; A)\;\mathtt{stk}\,([x]\,\Psi\, A)\\ \overset{\Psi}{=} \quad & ((\,\widehat{\rho}\,([x] +\!\!+ t)).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(x \cdot \Pi\, t)).\,\mathtt{snd}\; A)\;\mathtt{stk}\,((\,\widehat{\rho} t).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(\Pi\, t)).(\,\mathtt{dr}\,(x \cdot \Pi\, t)).\,\mathtt{snd}\; A)\\ \overset{\mathtt{stk}}{=} \quad & (\,\widehat{\rho}\,(s([x] +\!\!+ t)t)).\,\mathtt{vec}\;(\,\mathtt{tk}\,(x\Pi\, t)(\,\mathtt{snd}\; A) \;\; +\!\!+ \;\; (\,\mathtt{tk}\,(\Pi\, t)).(\,\mathtt{dr}\,(x \cdot \Pi\, t))(\,\mathtt{snd}\; A))\\ = \quad & (\,\widehat{\rho}\,([x+1] +\!\!+ t)).\,\mathtt{vec}\;(\,\mathtt{tk}\,(\Pi\, t + x \cdot \Pi\, t)(\,\mathtt{snd}\; A))\\ = \quad & (\,\widehat{\rho}\,([x+1] +\!\!+ t)).\,\mathtt{vec}\;(\,\mathtt{tk}\,((x+1) \cdot \Pi\, t)(\,\mathtt{snd}\; A))\end{aligned}$$

□

*Proof.* [stacking] Let $\rho\, A = [n] +\!\!+ t$ then

$$\begin{aligned} & (\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*(+\!\!+\,/_{[\,]}\,).\iota^*(\,\mathtt{tk}\,1)\rho\, A)_*\, A = \\ = \quad & (\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*.(+\!\!+\,/_{[\,]}\,)[\iota n])_*\, A\\ = \quad & (\,\mathtt{stk}\,/_\Theta\,).(\,\Psi\,^*.[\_]^*(\iota n))_*\, A\\ \overset{\text{lemma}}{=} \quad & (\,\widehat{\rho}\,([n] +\!\!+ t)).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(n \cdot \Pi\, t)).\,\mathtt{snd}\; A\\ = \quad & (\,\widehat{\rho}\,(\rho\, A)).\,\mathtt{vec}\;.(\,\mathtt{tk}\,(\#.\,\mathtt{snd}\; A)).\,\mathtt{snd}\; A\\ = \quad & (\,\widehat{\rho}\,(\rho\, A)).\,\mathtt{vec}\;(\,\mathtt{snd}\; A)\\ = \quad & A\end{aligned}$$

□

**13. Parallel programming and Second-Order Operators.** It is now a theme of parallel programming research that second-order operators provide language primitives to express communication structure [6, 7, 19, 29]. This is especially true of functional programs with arrays targeted at architectures whose topologies are meshes, hence are describable by array forms. We now introduce a method for recursive definitions of functions on arrays. The resulting functions can be interpreted as parallel algorithms for mesh-connected computers.

THEOREM 13.1 (RECURSION). *Given $t_\Theta \in T$, $h : S \to T$, and $G : T^* \to T$, any set of equations*

$$\begin{array}{rcl} F\Theta & = & t_\Theta \\ F(\,\mathtt{scl}\ a) & = & ha \\ \rho\, A \notin \{[0],[\,]\} \Rightarrow & & \\ FA & = & G.F^*.(\,\Psi^*.[\_]^*.(+\!\!+ /_{[\,]}\,).\iota^*.(\,\mathtt{tk}\,1).\rho\, A)_*\, A \end{array}$$

*defines a unique $F : S^\square \to T$.*

*Proof.* By induction on $\rho\, A$. If $\rho\, A \in \{[0],[\,]\}$, then the first two equations provide a unique value for $FA$. Otherwise, let $\rho\, A = [n] +\!\!+ f$, where $n \geq 2$. Then the list

$$(\,\Psi^*.[\_]^*.(+\!\!+ /_{[\,]}\,).\iota^*.(\,\mathtt{tk}\,1).\rho\, A)_*\, A = [[0]\,\Psi\, A, \ldots, [n-1]\,\Psi\, A]$$

consists of subarrays of $A$, all of shape $f$ (by proposition 8.10) and so by induction hypothesis all the entries of

$$F^*.(\,\Psi^*.[\_]^*.(+\!\!+ /_{[\,]}\,).\iota^*.(\,\mathtt{tk}\,1).\rho\, A)_*\, A = [F([0]\,\Psi\, A), \ldots, F([n-1]\,\Psi\, A)]$$

are well-defined. $\square$

Let $e : T$, $g : T \to T \to T$ and suppose $F_{g,e}$ is given the recursive definition where $t_\Theta = e$, $ha = gea$ and $G = g/_e$ :

(1) $$F_{g,e}A = g/_e\,[F([0]\,\Psi\, A), \ldots, F([n-1]\,\Psi\, A)]$$

Then we can prove by induction that $F_{g,e} = g/\!\!\!\square_e$ , i.e. $F_{g,e}A = g/_e\,(\,\mathtt{snd}\ A)$ if $(T, g, e)$ is a monoid. The induction step uses propositions 9.2 and 12.2.

Consider now the following operational interpretation of the recursive definition for $F_{g,e} : T^\square \to T$. Suppose the elements of $A \in T^\square$ are stored one at each node of a mesh-connected network of processing elements of dimensions $\rho\, A$, in the same order as they are listed in $\mathtt{snd}\, A$ (i.e. in lexicographic order of coordinates). For the base case of a scalar $A$, this corresponds to a processing element holding a single value. We can therefore assume by induction, that there exists a data-parallel algorithm for the mesh which computes the reduction $F_{g,e}A$ and leaves the resulting value on the lexicographically-first node. The inductive proof that this is possible simply requires the definition of a sequence of shift operations along the first dimension of $A$ to compute the list reduction given by the right expression of equation 1. The resulting algorithm is an elementary operation for mesh-connected SIMD computers: its execution time is $O(+/_0\,(\rho\, A))$. It is then possible to program, for example, a hypercube sum algorithm by transforming the list to be added into a hypercube-shaped array and then applying $F_{+,0}$:

$$F_{+,0}.(\,\widehat{\rho}\,[2, 2, \ldots, 2]).\,\mathtt{vec}\ \ : T^* \to T$$

Mapping to a different architecture form $f'$ is done by replacing $[2, 2, \ldots, 2]$ by $f'$ in the above expression.

Let again $e : T$, $g : T \to T \to T$ and suppose $H_{g,e}$ has the recursive definition where $t_\Theta = \Theta$, $ha = \ \mathtt{scl}\ (a)$, and $G = g'/_\Theta$ where

$$g'AB = A\,\mathtt{stk}\,((g(\mathrm{last}.\,\mathtt{snd}\ A))^\square B)$$

Here it is possible to prove that $H_{g,e}A = g\not\!\not\Box_e A$ if $(T, g, e)$ is a monoid. The most natural operational meaning for $H_{g,e}$ is a mesh-connected algorithm for scan where $g'(H_{g,e}A)(H_{g,e}B)$ distributes the function $g(\, g\not\Box_e\, A)$ over $(H_{g,e}B)$, thus maintaining the appropriate invariant. Unlike circuit or PRAM versions of this parallel prefix algorithm [21], $H_{g,e}$ contains all the required placement and communications information, assuming a virtual mesh-connected structure of processing elements. It is well-known that $\not\!\not\Box$ is the main parallel routine for a carry-save adder [21]. It is therefore possible to program a mesh adder algorithm via $H_{g,e}$.

The recursion theorem provides the outline of an abstract machine for implementing MOA operations, and algebraic identities a rich set of program transformations. The operations we chose to define here have a long history of applications in APL and are known to be useful program construction tools.

**14. Conclusions.** The development presented here is only one of many possible directions in which an algebra of arrays could be expanded. Its choice of functions was influenced by APL traditions and, more characteristic, by the goal of modeling parallel algorithms on distributed architectures whose topologies are product graphs. We have explored the consequences of the following informal hypotheses:

- Operations on arrays should be uniform across dimensions.
- Arrays are composed and decomposed in *slices* rather than *blocks.*

The first hypothesis is visible in the existence of a type of all arrays $T^{\Box}$ for a given base type $T$. MOA functions with array domains were defined to operate on the whole of $T^{\Box}$. For example, $l\,\Psi$ has type $T^{\Box} \rightarrow T^{\Box}$, regardless of the length of $l$ or the dimension of its array argument. It may be argued that there is no fundamental difference between this and a hierarchy of definitions over the range of `dim` , but we found our point of view useful in discovering patterns of behaviour which are dimension independent.

The second hypothesis is realised by the lexicographic ordering of content elements and the corresponding definition for $\Psi$ . A similar development is conceivable with contents in a generalisation of shuffled row-major order i.e. block order [25]. Although it is known to be useful in algorithm design shuffled ordering complicates definitions and is therefore less elementary. Loosely speaking, when programming with functional arrays lexicographic ordering leads to unidirectional dataflows while shuffled row-major ordering leads to recursive tree-shaped dataflows, for example the mesh implementation of PRAM algorithms described in [9].

It is necessary to evolve an axiomatic or categorical understanding of the algebra of arrays to explain and generalise its properties. Forthcoming work will be aimed at expressing MOA in constructive mathematics with the goal of computer-assisted theorem proving. We are currently developing a compiler for a dialect of ML with arrays to generate MIMD code for transputer arrays.

**15. Acknowledgements.** MOA was inspired by APL, some of its functions being adapted from it, but the goals of both formalisms are different and there are many fundamental differences (strong typing, no array overloading of arithmetic functions, empty array, congruence of forms etc.). We borrowed several features of our presentation from the Bird-Meertens formalism [5, 4] (some syntax, curried functions, list homomorphisms, equational proofs etc.).

There has been several formal definitions of arrays but to our knowledge none with dimension-independent operators. Early work on the verification of APL programs can be found in [8]. A few references to other formal definitions of arrays and their algorithms are [26, 23, 17, 16, 10, 5, 22]. This list is not exhaustive.

What follows is a short list of sources which influenced the current version of MOA. We apologise for any incompleteness. The design for an APL machine of Phil Abrams [1] introduced the key concept of optimising memory access through manipulation of array forms. Other work on APL systems can be found in [13, 11]. Tu and Perlis [32] showed the usefulness of arrays in $\lambda$-expressions. Klaus Berkling proposed and encouraged the development of MOA. Joseph Mann contributed to the present state of the formalism, in particular the flat content list. David Wise raised the issue of redundant occurrences of 0 and 1 in forms. Richard Bird, Geraint Jones, Bill McColl, Robin Milner, Christine Paulin and Michel Boyer made other useful comments. Christian Foisy contributed to some of the proofs in later sections. Lenore Mullin wishes to thank Kenneth Iverson for teaching her to think in arrays and Garth Foster for introducing her to Abrams' work. Gaétan Hains thanks Michel Boyer for his witty comments and encouragement.

## REFERENCES

[1] P. S. Abrams, *An APL Machine*, PhD dissertation, Stanford University, 1970.

[2] J.-P. Banâtre, A. Coutant, and D. LeMétayer, *A parallel machine for multiset transformation and its programming style*, Future Generations Computer Systems, 4 (1988), pp. 133–144.

[3] D. Benanav, D. Kapur, and P. Narendran, *Complexity of matching problems*, in First International Conference on Rewrite Techniques and Applications, Dijon, France, May 1985.

[4] R. Bird, *A calculus of functions for program derivation*, in Research Topics in Functional Programming, D. A. Turner, ed., Addison-Wesley, 1990.

[5] R. S. Bird, *Lectures on constructive functional programming*, in Constructive Methods in Computing Science, M. Broy, ed., no. 55 in NATO ASI, Marktoberdorf, BRD, 1989, Springer.

[6] G. Blelloch, *Scans as primitive parallel operations*, in International Conference on Parallel Processing, S. K. Sahni, ed., IEEE Computer Society, August 1987, pp. 355–362.

[7] G. E. Blelloch, *Vector Models for Data-Parallel Computing*, MIT Press, 1990.

[8] S. L. Gerhart, *Verification of APL Programs*, PhD thesis, Carnegie Mellon University, Computer Science Department, 1972.

[9] A. M. Gibbons and Y. N. Srikant, *A class of problems efficiently solvable on mesh-connected computers including dynamic expression evaluation*, Information Processing Letters, (1989), pp. 305–311.

[10] J. Glasgow, M. Jenkins, C. McCrosky, and H. Meijer, *Expressing parallel algorithms in Nial*, Parallel Computing, (1989), pp. 331–347.

[11] L. J. Guibas and D. K. Wyatt, *Compilation and delayed evaluation in APL*, in Fifth Annual ACM Symposium on the Principles of Programming Languages, 1978.

[12] G. Hains, *La complexité du filtrage associatif-commutatif*, Comptes Rendus de l'Académie des Sciences, Paris, t.311, Série I (1990), pp. 741–744.

[13] A. Hassitt and L. E. Lyon, *Efficient evaluation of array subscripts*, IBM Journal of R& D, (1972).

[14] P. Hudak, *Conception, evolution and application of functional programming languages*, Computing Surveys, 21 (1989).

[15] B. Jayaraman and D. A. Plaisted, *Functional programming with sets*, in Functional Programming Languages and Computer Architecture, G. Kahn, ed., no. 274 in Lecture Notes in Computer Science, Springer, 1987.

[16] J. Jeuring, *The derivation of hierarchies of algorithms on matrices.* CWI, Centrum voor Wiskunde en Informatica, Amsterdam, December 1990.

[17] ———, *A hierarchy of algorithms for pattern matching on matrices.* CWI, Centrum voor Wiskunde en Informatica, Amsterdam, November 1990.

[18] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, North-Holland, 1990.

[19] P. Kelly, *Functional Programming for Loosely-Coupled Multiprocessors*, Research Monographs in Parallel and Distributed Computing, Pitman – MIT Press, 1989.

[20] P. M. Kogge and H. S. Stone, *A parallel algorithm for the efficient solution of a general class of recurrence equations*, IEEE Tr. on Computers, C22 (1973), pp. 786–793.

[21] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, Journal of the ACM, 27 (1980),

pp. 831–838.

[22] C. McCrosky, *Realizing the parallelism of array-based computation*, Parallel Computing, 10 (1989), pp. 29–43.

[23] T. More, *Notes on the diagrams, logic and operations of array theory*, in Structures and Operations in Engineering and Management Systems, Bjorke and Franksen, eds., Trondheim, Norvège, 1981, Tapir Pub.

[24] L. M. R. Mullin, *A Mathematics of Arrays*, doctoral dissertation, Syracuse University, Syracuse, N.Y., December 1988.

[25] D. Nassimi and S. Sahni, *Data broadcasting in SIMD computers*, IEEE transactions on computers, c-30 (1981), pp. 101–107.

[26] J. C. Reynolds, *Reasoning about arrays*, Communications of the ACM, 22 (1979).

[27] J. Siekmann, *Universal unification*, in $7^{th}$ International Conference on Automated Deduction, Shostak, ed., no. 170 in Lecture Notes in Computer Science, Springer, 1984.

[28] ———, *An introduction to unification theory*, in Formal Techniques in Artificial Intelligence, R. B. Banerji, ed., North-Holland, 1990.

[29] D. Skillicorn, *Architecture independent parallel computation*, IEEE Computer, 23 (1990).

[30] L. Tao, *Mapping Parallel Programs onto Parallel Systems with Torus and Mesh Based Communication Structures*, PhD thesis, University of Pennsylvania, Dept. of Computer and Information Science, Philadelphia, Pennsylvania 19104, 1988. 137 Pages.

[31] L. Tao and E. Ma, *Simulating parallel neighboring communications among square meshes and square toruses*, Journal of Supercomputing, (1991).

[32] H. Tu and A. J. Perlis, *FAC: A functional APL language*, IEEE Software, (1986).

[33] L. G. Valiant, *Optimally universal parallel computers*, in Opportunities and Constraints of Parallel Computing, J. L. C. Sanz, ed., Springer, 1989.

[34] ———, *Optimally universal parallel computers*, in Scientific Applications of Multiprocessors, R. Elliott and C. A. R. Hoare, eds., Prentice-Hall, 1989, pp. 17–20.

[35] P. M. B. Vitányi, *Locality, communication, and interconnect length in multicomputers*, SIAM Journal on Computing, 17 (1988), pp. 659–672.