Laboratoire de l'Informatique du Parallélisme

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

A Categorical Model of Array Domains

Gaétan Hains and John Mullins

Décembre 1994

Research Report N° 94-43

Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

A Categorical Model of Array Domains

Gaétan Hains and John Mullins

Décembre 1994

Abstract

We apply the theory of generalised concrete data structures (or gCDSs) to construct a cartesian closed category of concrete array structures with explicit data layout. The technical novelty is the array gCDS preserved by exponentiation whose isomorphisms relate higher-order objects to their local parts. This work is part of our search of semantic foundations for data-parallel functional programming.

Keywords: parallel programming; functional languages; distributed arrays; denotational semantics; concrete data structures.

Résumé

Nous construisons une catégorie cartésienne fermée de structures de données concrètes explicitement réparties. Les principaux isomorphismes font correspondre tout objet à ses parties locales. Ce travail s'inscrit dans notre recherche de fondements sémantiques pour la programmation fonctionnelle data-parallèle.

Mots-clés: programmation parallèle ; langages fonctionnels; tableaux répartis; sémantique dénotationnelle; structures concrètes

A Categorical Model of Array Domains

Gaétan Hains*† and John Mullins‡§

December 27, 1994

Abstract

We apply the theory of generalised concrete data structures (or gCDSs) to construct a cartesian closed category of concrete array structures with explicit data layout. The technical novelty is the array gCDS preserved by exponentiation whose isomorphisms relate higher-order objects to their local parts. This work is part of our search of semantic foundations for data-parallel functional programming.

Keywords: parallel programming; functional languages; distributed arrays; denotational semantics; concrete data structures.

Contents

*LIP, ENS Lyon, 46 Allée d'Italie, F-69364 Lyon Cédex 07, France. On leave from Université de Montréal.
†First author supported by a visiting position of ENS-Lyon and a grant by FCAR-GRIAO (Québec).
‡CNRS-CRIN, Bâtiment LORIA, B.P. 239, F-54506, Vandoeuvre-les-Nancy Cédex, France. On leave from University of Ottawa.
§Second author supported by a visiting position of CNRS-CRIN and a grant by NSERC (Canada).

1 Introduction

Concrete data structures (CDSs) [3] and their abstract counterparts concrete domains [5] occur in the semantic study of sequential functions, deterministic parallel algorithms and other semantic notions. These structures allows the construction of several Cartesian closed categories (CCCs), standard models for typed functional languages.

In this report we apply Brookes and Geva's theory of generalised CDSs (or gCDSs) [1] and construct a CCC of domains whose objects are the sets of states of *array structures* whose cells, values and events are labelled by network addresses called *indices*. The construction is such that communication along the network's edges is represented by the enabling relation of gCDSs. The present model has the advantage of ensuring consistently complete domains.

This construction is intended as a model for data-parallel objects and is part of an exploration of possible foundations for the notion of network address (or data placement, or task allocation) in data-parallel functional languages. Its target application is the description of such languages' parallel primitives in an "internal" way.

2 Concrete data structures

We will write the isomorphism relation as $\equiv$. The space of continuous functions from A to B (ordered pointwise) is written $A \to B$ or sometimes $[A \to B]$.

2.1 The discrete case

This subsection summarises basic notions about concrete data structures. We borrow the presentation from [1] but first specialise it to classical (discrete) CDSs whose cells are not ordered. The CDSs were used by Berry and Curien to explore semantic models of sequential computation. Brookes and Geva have defined generalised CDS (or gCDS) and parallel algorithms over them. Our purpose is to apply the gCDSs but we first present CDSs to illustrate the theory in its simpler form.

A *concrete data structure* or CDS is a tuple $(C, V, E, \vdash)$ where

- C is a countable set of *cells*.
- V is a countable set of *values*.
- $E \subseteq C \times V$ is a set of *events*. An event (c, v) is often written cv.
- $\vdash$, the *enabling* relation is defined between finite sets of events and cells. The enabling relation induces a *precedence* relation on cells: $c \ll c'$ iff $\exists y, v.\ y \cup \{cv\} \vdash c'$. This enabling relation must be well-founded.

A cell c is called *initial* if it is enabled by the empty set of events (this is sometimes written $\vdash c$). A cell c is said to be *filled* in a set of events y if $\exists v.\ cv \in y$. We write $F(y)$ for the set of cells filled in y. If $y \vdash c$, then y is said to be an *enabling* of c and c is said to have an enabling in any superset y' of y, written $y \vdash_{y'} c$. Let $E(y)$ be the set of cells enabled in y, and call $A(y) = E(y) - F(y)$ the set of cells *accessible* from y. Let M, M', N denote CDSs from now on.

A *state* of M is a set of events $x \subseteq E_M$ which is functional and safe:

- $cv_1, cv_2 \in x \;\Rightarrow\; v_1 = v_2$.
- $c \in F(x) \;\Rightarrow\; \exists y \subset x.\ y \vdash_x c$

The second property (safety) is the requirement that each of the state's cells should have a proof in it: a derivation through $\vdash$ (or more accurately, $\ll$) starting with initial cells as axioms. Define $\mathcal{D}(M)$ to be the poset of states of CDS M, ordered by set inclusion.

The following examples give a glimpse of how concrete structures can be applied to the description of typed functional languages (see Curien's book [3] for a complete exposition of both the theory and its application). Let $\texttt{Bool} = (\{B\}, \{T, F\}, \{BT, BF\}, \vdash)$ where B is initial. Then $\mathcal{D}(\texttt{Bool})$ is $\{\emptyset, \{BT\}, \{BF\}\}$ and represents the flat domain of boolean values. The CDS $\texttt{Nat} = (\{N\}, N, \{Nn \mid n \in N\}, \vdash)$ where N is initial has states $\mathcal{D}(\texttt{Nat}) = \{\emptyset, \{N0\}, \{N1\}, \ldots\}$ and represents the flat domain of naturals. The following structure

$$(\{S, B, N\}, \{L, R\} \cup V_{\texttt{Bool}} \cup V_{\texttt{Nat}}, \{SL, SR\} \cup E_{\texttt{Bool}} \cup E_{\texttt{Nat}}, \vdash)$$

where the enablings are $\vdash S$, $SL \vdash B$, and $SR \vdash N$, has a state domain which encodes the sum of Bool and Nat (end of example).

The following is a simple structure which will be used later.

$$\texttt{Vnat} = (N, \{*\}, \{n* \mid n \in N\}, \vdash) \text{ where } \vdash 0 \text{ and } \{k*\} \vdash (k+1).$$

Its domain of states is isomorphic to the vertical ordering of the natural numbers with a point at infinity. It illustrates how the enabling relation defines a concept of trace (in fact it plays the role of trace index in the Brookes-Geva theory of parallel algorithms). Our definition of array structures extends this meaning to a context where enablings may correspond either to local computation or to communications.

The posets obtained as the states of CDSs are called *concrete domains* (or CDs) and have the following properties.

Proposition 1 (Brookes and Geva [1]) *CDs are (consistently complete) Scott domains where $\perp$ is the empty set, lowest upper bounds are given by unions and the compacts are the finite states.*

Recall from the theory of Scott domains (c.f. [4]) that when D, E are consistently complete domains then so is $[D \to E]$. Moreover if D and E have so-called effective presentations, then so does $[D \to E]$. These are necessary properties for the constitution of a CCC and its use as a computational model, but in the present case they are not sufficient. Berry and Curien have shown that none of the continuous, stable or sequential function space constructions preserve CDSs. In other words $[\mathcal{D}(M) \to \mathcal{D}(N)]$ is a consistently complete domain but is not isomorphic to a concrete domain.

The same holds for the domains of stable or sequential functions. As a result they successfully explored a category of CDs whose arrows are sequential *algorithms*. It turns out that our definition of array structures is not suitable for this "sequential" theory, array structures being inherently parallel (non-deterministic in the terminology of [3]). And since our target application is the description of data-parallel languages, we apply instead the theory of genralised concrete data structures which we now summarise.

2.2 Generalised concrete data structures

A generalised CDS is equipped with a partial order $\leq$ on its cells and must satisfy the following additional properties: its set of events and its enabling relation must be upwards-closed with respect to cell ordering. Namely

1. $cv \in E,\ c \leq c' \ \Rightarrow\ c'v \in E$

2. $y \vdash c,\ c \leq c' \ \Rightarrow\ y \vdash c'$.

As before the precedence relation on cells must be well-founded. A new requirement in the generalised definition is that states over the gCDS must be upwards-closed with respect to cell ordering:

$$cv \in x,\ c \leq c' \ \Rightarrow\ c'v \in x$$

Any CDS is a gCDS with discrete order on its cells. The domains built from gCDS states are called *generalised* concrete domains (gCD) and satisfy proposition 1.

The following is Brookes and Geva's construction of a CCC of gCDS's and continuous functions called `gCDScont` [1]. Its objects are the gCDS and the arrows are the continuous functions between corresponding gCDs.

Let $c.i$ denote a pair (c, i) where i is an integer tag and extend this notation to sets of cells, events etc. The product $M_1 \times M_2 = (C, \leq, V, E, \vdash)$ of two gCDSs $M_i = (C_i, \leq_i, V_i, E_i, \vdash_i)$ is defined by a pointwise construction on pairs. Suffixes of the form $.k$ applied to a set are understood to distribute onto its elements.

- $C = C_1.1 \cup C_2.2$
- $c.i \leq c'.i'$ if and only if $c \leq_i c'$ and $i = i'$.
- $V = V_1 \cup V_2$
- $E = E_1.1 \cup E_2.2$
- $y.i \vdash c.i$ if and only if $y \vdash_i c$.

Proposition 2 (Brookes and Geva [1]) *The product preserves gCDSs and is a categorical product in* `gCDScont` *with the following pairing and projections:*

$$\begin{array}{rcl} \langle x_1, x_2 \rangle & = & x_1.1 \cup x_2.2 \\ \pi_i(x) & = & \{cv \mid c.i\, v \in x\}. \end{array}$$

$\mathcal{D}(M_1) \times \mathcal{D}(M_2)$ is isomorphic to $\mathcal{D}(M_1 \times M_2)$.

Let two gCDSs M, M' be given and let us call them temporarily the source structure and the target structure. The exponential gCDS

$$M \rightarrow M' = (C, \leq, V, E, \vdash)$$

is defined as follows

- $C = \mathcal{D}_{fin}(M) \times C_{M'}$, where $\mathcal{D}_{fin}(M)$ is the set of finite states of M ordered by inclusion. We will write xc' as an abbreviation for (x, c'). A cell in the exponential is built from a finite state in the source structure and a cell in the target structure.
- $\leq\, = \,\subseteq \times \leq_{M'}$. The former are ordered by inclusion and the latter retain their order relation.
- $V = V_{M'}$. A value of the exponential is a value of the target structure.
- $E = \{xc'v' \in C \times V \mid c'v' \in E_{M'}\}$. Strictly speaking, an event in the exponential structure is a pair (xc', v'). But viewing it instead as the pair $(x, c'v')$ highlights its intended meaning. It associates an event $c'v'$ of the target structure to a finite state x in the source structure. In other words it is a finitary piece of a map from states to states.

- $\{x_j c'_j v'_j \mid 1 \leq j \leq l\} \vdash xc'$ if and only if $\{c'_j v'_j \mid 1 \leq j \leq l\} \vdash_{M'} c'$ and $x_j \subseteq x$ for all j. A cell of the exponential xc' is enabled by a set of events exactly when the source-states parts of those events are subsets of x and the target-events parts enable c'.

The exponential preserves gCDSs.

Proposition 3 (Brookes and Geva [1]) *An exponential domain is isomorphic to its space of continuous functions ordered pointwise,*

$$\mathcal{D}(M \to N) \equiv [\mathcal{D}(M) \to \mathcal{D}(N)],$$

so that continuous functions between the two gCDS M and N are equivalent to states of $\mathcal{D}(M \to N)$. The isomorphism and its inverse are given by:

$$a \in \mathcal{D}(M \to N) \quad \mapsto \quad \lambda z \in \mathcal{D}(M).\, \{c'v' \mid \exists x \subseteq z.\ xc'v' \in a\} \tag{1}$$

$$f \in [\mathcal{D}(M) \to \mathcal{D}(N)] \quad \mapsto \quad \{xc'v' \in E \mid c'v' \in f(x)\} \tag{2}$$

The two isomorphisms allows us to interchange continuous functions and states of the exponential structure. Moreover, application and curryfication satisfy both halves of the definition of an exponentiation in `gCDScont` [7].

Corollary 1 *`gCDScont` is a CCC.*

As a side remark about the relationship of gCDSs to CDS, the above exponentiation does not preserve discreteness of CDSs. In general, when M, M' are discrete, $\mathcal{D}_{fin}(M)$ carries its non-trivial cell order ($\subseteq$) into the ordering of ($M \to M'$)'s cells. For this reason the theory of gCDS is not actually an extension of the theory of CDS; the objects are extensions but the morphisms are not: the closure by continuous functions generates a CCC from generalised concrete domains but not from (discrete) concrete domains.

3 Array structures

Given a CDS whose states will be our singleton arrays (or *scalars*), we construct an array CDS by replicating the cells over the nodes or *indices* of a graph. This graph indirectly defines the enabling relation as explained below. Array indices thus represent adresses in a static ("physical") multiprocessor network.

In the remainder we assume the existence of a fixed countable directed graph (I, L) whose nodes $\vec{\imath} \in I$ will be called indices and whose edges $l = (\vec{\imath}, \vec{\jmath}) \in L$ will be called channels or *links*.

Let $M = (C_0, \leq_0, V_0, E_0, \vdash_0)$ be a given gCDS. We now define $M^\Box$ the array data structure or *array structure* over M and show that it is a gCDS. The components of $M^\Box = (C, \leq, V, E, \vdash)$ are defined as follows.

- $C = I \times C_0$ is countable because both I and C_0 are. A cell $(\vec{\imath}, c)$ or $\vec{\imath}c$ in the array structure is said to be *located* at (*location*) $\vec{\imath}$. Cells are ordered locally: $\vec{\imath}c \leq \vec{\jmath}c'$ if and only if $\vec{\imath} = \vec{\jmath}$ and $c \leq_0 c'$.

- $V = V_0$ is countable by hypothesis.

- $E = I \times E_0$ the possible events are the localisations of possible scalar events. The type of E is correct since $E_0 \subseteq C_0 \times V_0$ and so $E \subseteq I \times C_0 \times V_0 = C \times V$.

- The enabling relation $\vdash$ is between $\mathcal{P}_{fin}(I \times E_0)$ and $I \times C_0$. There are two types of enablings, local or through a link:
  - $\{\vec{\imath}c_1v_1, \ldots, \vec{\imath}c_kv_k\} \vdash \vec{\imath}c$ when $\{c_1v_1, \ldots, c_kv_k\} \vdash_0 c$.
  - $\{\vec{\jmath}c_1v_1, \ldots, \vec{\jmath}c_kv_k\} \vdash \vec{\imath}c$ when $(\vec{\jmath}, \vec{\imath}) \in L$ and $\{c_1v_1, \ldots, c_kv_k\} \vdash_0 c$.

The first type of enabling allows $M^\Box$ to recover a copy of the enabling relation of M at any location $\vec{\imath}$; events located at a common site enable non-initial cells at the same site according to $\vdash_0$. The second condition defines the expansion of states to new locations; a cell located at $\vec{\imath}$ is enabled by a set of events located at a neighbouring index of $\vec{\jmath}$. As a result, $\vdash_0$-initial cells remain initial "everywhere" in $M^\Box$.

Because M is a gCDS, it follows that E and $\vdash$ are upwards closed with respect to $\leq$. The following property of the enabling relation must also be satisfied to make $M^\Box$ a gCDS.

Lemma 1 $\ll$ *is well-founded.*

Proof: By considering both types of enablings in $\vdash$ we find that, $\vec{\jmath}c' \ll \vec{\imath}c$ only if $c' \ll_0 c$. Therefore a descending chain $\vec{\imath}_1c_1 \gg \vec{\imath}_2c_2 \gg \ldots$ corresponds to a descending chain in M: $c_1 \gg_0 c_2 \gg_0 \ldots$. But since M is a gCDS, its $\ll_0$ is well-founded and such chains must be finite. $\Box$

As a result the set $\mathcal{D}(M^\Box)$ of states of an array structure is a concrete domain and, by proposition 1, a consistently complete Scott domain. We will call it an *array domain* and its states will be called *arrays* over M.

Two last remarks about $M^\Box$. Because a cell can be enabled either locally or remotely, enablings are not unique even when they were so in M. Also, because the cells of $t \in \mathcal{D}(M^\Box)$ may have been enabled remotely, the set $t\vec{\imath} = t \cap (\{\vec{\imath}\} \times C_0 \times V_0)$ is not in general a state of $\mathcal{D}(M)$.

Up to now we know that $\mathcal{D}(M^\Box)$ is a Scott domain for inclusion. It can be verified that the compacts of $\mathcal{D}(M^\Box)$ are the finite arrays.

4 A CCC of array domains

We now prove that array structures form a category with a terminal object, finite products and exponentiations.

4.1 Composition and terminal object

Let M_1, M_2, M_3 be CDSs. Then the identity transformations on $\mathcal{D}(M_i^\Box)$ are continuous transformations (with respect to $\subseteq$). If $f \in [\mathcal{D}(M_1^\Box) \to \mathcal{D}(M_2^\Box)]$ and $g \in [\mathcal{D}(M_2^\Box) \to \mathcal{D}(M_3^\Box)]$ then $g \circ f \in [\mathcal{D}(M_1^\Box) \to \mathcal{D}(M_3^\Box)]$. We may therefore define the category `ADScont` with array data structures $M^\Box$ as objects, continuous transformations between their array domains $\mathcal{D}(M^\Box)$ as morphisms, function composition as composition and identity transformations as identity morphisms. This is no surprise as `ADScont` is a subcategory of `gCDScont` .

Let `Null` be the CDS $(\emptyset, \emptyset, \emptyset, \emptyset)$ whose only state is $\emptyset$. The array structure $\texttt{Null}^\Box$ is also $(\emptyset, \emptyset, \emptyset, \emptyset)$ and therefore the only element of $\mathcal{D}(\texttt{Null}^\Box) = \mathcal{D}(\texttt{Null})$ is the empty state. As a result there is a unique continuous function $\emptyset \in [\mathcal{D}(M^\Box) \to \mathcal{D}(\texttt{Null})]$ and so $\texttt{Null}^\Box$ is a terminal object in ADScont.

4.2 Product: pairs of arrays

The product of array structures is a special case of the product of CDSs (defined in subsection 2.2). A pair of arrays corresponds by geometrical superposition to an array of pairs. Let $M_k^\square = (I \times C_k, V_k, I \times E_k, \vdash_k{}^\square)$ for $k = 1, 2$ be two array structures over $M_k = (C_k, V_k, E_k, \vdash_k)$. The array product structure $M_1^\square \times M_2^\square = (I, V, E, \vdash)$ is determined by the definition of product.

- $I = (I \times C_1).1 \cup (I \times C_2).2$
- $V = V_1 \cup V_2$
- $E = (I \times E_1).1 \cup (I \times E_2).2$
- $\{\vec{\imath}_l e_l.k_l\}_l \vdash \vec{\imath}c.k$ if and only if $\forall l.\ k_l = k$ and $\{\vec{\imath}_l e_l\}_l \vdash_k \vec{\imath}c$. In other words the two enabling relations are superimposed without interaction.

The array construction preserves finite products because there is a natural correspondence between arrays of pairs and pairs of arrays.

Lemma 2 *The structures $(M_1 \times M_2)^\square$ and $M_1{}^\square \times M_2{}^\square$ are isomorphic (i.e. their state domains are).*

Proof: Consider a state x of $(M_1 \times M_2)^\square$ and define

$$\mathtt{split}_{M_1,M_2} x = (x_1, x_2) \quad \text{where} \quad x_i = \{e \mid e.i \in x\}$$

Clearly x_1 and x_2 are sets of events in $M_1{}^\square$ and $M_2{}^\square$ respectively. It is straightforward to verify that both are functional (because x is), that all their events have enablings (by definition of the product enabling) and that they are upwards-closed for the cell order (componentwise in $M_1{}^\square \times M_2{}^\square$). As a result x_i is a state of $M_i{}^\square$. The inverse of $\mathtt{split}_{M_1,M_2}$ is $\mathtt{merge}_{M_1,M_2}$, it reconstructs x:

$$\mathtt{merge}(x_1, x_2) = x_1.1 \cup x_2.2.$$

The correspondence is bijective and preserves unions. □
Since the product is a categorical product in `gCDScont` , and since it preserves array domains, it is also a categorical product in ADScont.

4.3 Exponential domains: array transformations

Here we show that the exponential operator preserves array structures, thus completing the proof that our category is cartesian closed.

Proposition 4 *For M, N generalised concrete structures, $(M^\square \to N^\square)$ and $(M^\square \to N)^\square$ are isomorphic.*

Proof: By applying the definitions of exponential and array structures, we will verify that both gCDS have the same cells, events and enablings. To do this let us consider a typical structure $(C, \leq, V, E, \vdash)$ of $(M^\square \to N^\square)$. We will give subscripts to the various sets involved according to the type of structure to which they belong. For example, C_N will denote the cells of N.

First of all, the cells are the same up to a permutation of their parts.

$$\begin{aligned} C &= \mathcal{D}_{fin}(M^\square) \times C_{N^\square} = \mathcal{D}_{fin}(M^\square) \times I \times C_N \\ &\equiv I \times \mathcal{D}_{fin}(M^\square) \times C_N = I \times C_{M^\square \to N} = C_{(M^\square \to N)^\square} \end{aligned}$$

The cell orderings are equal up to the same permutation.

$$\begin{array}{rcl} \leq & = & \subseteq \times \leq_{N^\square} = \subseteq \times \mathrm{id}_I \times \leq_N \\ & \equiv & \mathrm{id}_I \times \subseteq \times \leq_N = \mathrm{id}_I \times \leq_{M^\square \to N} = \leq_{(M^\square \to N)^\square} \end{array}$$

The values are the same.

$$V \quad = \quad V_{N^\square} = V_N = V_{M^\square \to N} = V_{(M^\square \to N)^\square}$$

We now use the following compact forms of the definitions of event sets

$$\begin{array}{rcl} E_{M \to N} & = & \mathcal{D}_{fin}(M) \times E_N \\ E_{M^\square} & = & I \times E_M \end{array}$$

to verify the equivalence of events, using the same permutation of parts as before.

$$\begin{array}{rcl} E_{M^\square \to N^\square} & = & \mathcal{D}_{fin}(M^\square) \times E_{N^\square} \\ & = & \mathcal{D}_{fin}(M^\square) \times I \times E_N \\ & \equiv & I \times \mathcal{D}_{fin}(M^\square) \times E_N \\ & = & I \times E_{M^\square \to N} = E_{(M^\square \to N)^\square} \end{array}$$

Recall now the definitions of activations for the exponential and array structures (here L^0 is the identity graph on I):

$$\begin{array}{rcl} \{x_j e_j \mid j = 1, \ldots n\} & \vdash_{M \to N} & xc \\ & \text{iff} & \{e_j \mid j = 1, \ldots n\} \vdash_N c \\ & & \text{and} \\ & & \forall j.\ x_j \subseteq x. \end{array}$$

and

$$\begin{array}{rcl} \{\vec{\jmath} e_j \mid j = 1, \ldots n\} & \vdash_{M^\square} & \vec{\imath} c \\ & \text{iff} & \{e_j \mid j = 1, \ldots n\} \vdash_M c \\ & & \text{and} \\ & & (\vec{\jmath}, \vec{\imath}) \in L \cup L^0 \end{array}$$

Now we verify that the enablings are the same, up to the permutation used for cells and events.

$$\begin{array}{rcl} \vdash_{M^\square \to N^\square} & = & \{(\{x_j \vec{\jmath} e_j\}_j, x \vec{\imath} c) \mid \{\vec{\jmath} e_j\}_j \vdash_{N^\square} \vec{\imath} c \text{ and } \forall j.\ x_j \subseteq x\} \\ & = & \{(\{x_j \vec{\jmath} e_j\}_j, x \vec{\imath} c) \mid \{e_j\}_j \vdash_N c \text{ and } (\vec{\jmath}, \vec{\imath}) \in L \cup L^0 \text{ and } \forall j.\ x_j \subseteq x\} \\ & \equiv & \{(\{\vec{\jmath} x_j e_j\}_j, \vec{\imath} x c) \mid \{e_j\}_j \vdash_N c \text{ and } \forall j.\ x_j \subseteq x\} \text{ and } (\vec{\jmath}, \vec{\imath}) \in L \cup L^0\} \\ & = & \{(\{\vec{\jmath} x_j e_j\}_j, \vec{\imath} x c) \mid \{x_j e_j\}_j \vdash_{M^\square \to N} c \text{ and } (\vec{\jmath}, \vec{\imath}) \in L \cup L^0\} \\ & = & \{(\{\vec{\jmath} x_j e_j\}_j, \vec{\imath} x c) \mid \{\vec{\jmath} x_j e_j\}_j \vdash_{(M^\square \to N)^\square} \vec{\imath} x c\} \\ & = & \vdash_{(M^\square \to N)^\square} \end{array}$$

$\square$

The isomorphism and its inverse are implicit in the above transformations. They simply interchange the role of indices and other parts of the structure. We will call them *localisation* and *globalisation*:

$$\begin{array}{l} \mathtt{loc} : \mathcal{D}(M^\square \to N^\square) \to \mathcal{D}((M^\square \to N)^\square) \\ \mathtt{loc}\, a = \{\vec{\imath} x e \mid x \vec{\imath} e \in a\} \\ \mathtt{glob} : \mathcal{D}((M^\square \to N)^\square) \to \mathcal{D}(M^\square \to N^\square) \\ \mathtt{glob}\, t = \{x \vec{\imath} e \mid \vec{\imath} x e \in t\} \end{array}$$

Both are continous transformations. In concrete terms, `loc` takes a continuous array transform and decomposes it into an array of scalar functionals. The inverse operation is to take the array of functionals and to return the transformation which applies every element of it to an argument array. We can now state that

Corollary 2 *ADScont is a CCC.*

Proof: `ADScont` is closed for (the terminal object and) the product and exponentiation of its enclosing category `gCDScont`. Moreover, in `gCDScont` application and curryfication satisfy the axioms for being a CCC. □

5 Conclusion

We have constructed a CCC of array structures which share a fixed index space, thus introducing the notion of physical placement into a functional setting. The semantics of several data-parallel functional languages like Crystal [2], Alpha [6, 8] and PEI [9] is based on data-fields or arrays. We hope that our future investigation can relate their semantics to considerations of communication and lead to higher-order versions of such languages. The first step in that direction will be to apply the category of array domains to the definition of a lambda-calculus.

Acknowledgements: G.H. would like to thank Gil Utard for his comments on this work. We thank Luc Bougé and David Cachera for proofreading the manuscript.

References

[1] S. Brookes and S. Geva. Continuous functions and parallel algorithms on concrete data structures. In *MFPS'91*, L.N.C.S. Springer, 1991.

[2] M. Chen, Y.-Il Choo, and J. Li. Crystal: Theory and pragmatics of generating efficient parallel code. In B. K. Szymanski, editor, *Parallel Functional Languages and Compilers*, chapter 7. ACM Press, 1991.

[3] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Birkhäuser, Boston, second edition, 1993.

[4] C. A. Gunter and D. S. Scott. Semantic domains. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science.* North-Holland, MIT-Press, 1990.

[5] G. Kahn and G. Plotkin. Domaines concrets. Rapport 336, INRIA-LABORIA, 1978.

[6] C. Mauras. Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones. Thèse de l'Université de Rennes 1, IFSIC, Décembre 1989.

[7] B. C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[8] H. Le Verge. Reduction operators in ALPHA. In D. Etiemble and J.-C. Syre, editors, *PARLE-92*, number 605 in Lecture Notes in Computer Science, Paris, France, June 1992. Springer.

[9] E. Violard and G.-R. Perrin. PEI: A language and its refinement calculus for parallel programming. *Parallel Computing*, 18(10):1167–1184, October 1992.