Proc. Estonian Acad. Sci. Phys. Math., 2003, **52**, 4, 337–355
https://doi.org/10.3176/phys.math.2003.4.01

On the category of data dependency algebras and embeddings

Alexa Anderlik and Magne Haveraaen

Department of Informatics, University of Bergen, P.O. Box 7800, N-5020 Bergen, Norway; {Alexa.Anderlik,Magne.Haveraaen}@ii.uib.no

Received 7 April 2003, in revised form 27 June 2003

Abstract. Programming a parallel computer is basically embedding the data dependency pattern of the program into the space-time pattern of the computer. In general this is a technically demanding task. By considering these patterns as algebraic structures – data dependency algebras and space-time algebras (a special case of data dependency algebras), we get a means for doing formal reasoning about the embedding problem, detached from concrete programs and machines. In this paper we show that it is possible to factorize data dependency algebras so that the embedding of a program into a parallel computer may be broken down to embeddings of simpler data dependency algebras into simpler space-time algebras. These simpler embeddings are then automatically combined to form a full embedding of the parallel program on the machine.

Key words: data dependency algebras, space-time embeddings, modularity, parallel programming, category of data dependency algebras.

1. INTRODUCTION

There are many approaches to programming high performance computers (HPCs). Most HPCs are parallel in one way or another, ranging from loosely coupled networks of independent computers to specialized vector processors. This spans from distributed memory machines, where data may have to be transmitted across some communication network to be accessed by the appropriate processor, to shared memory machines where all processors may directly access each data item.

Programming these machines is difficult, and several models exist. The simplest to reason about is the single program, multiple data model (SPMD) model, as embodied in HPF [1] or in many functional languages (see [2] for an overview).

More complicated to program are variations of the multiple program, multiple data model (MPMD). Such programs are prone to errors such as deadlock in addition to the normal sequential programming errors. MPMD programming is achieved by adding communication primitives, such as MPI [3,4], to a sequential language.

The concept of data dependencies is central to the efficient use of parallelism. The dependency patterns represent constraints defined by the algorithms we want to run on computers. Any execution of a program must obey its dependency pattern. This is trivially achieved on sequential machines. On a parallel machine we want to distribute the program in order to utilize the processing resources as much as possible. The task can then be seen as embedding the program's data dependency onto the hardware's communication structure so that we utilize the computing resources as much as possible.

In [5,6] we suggested abstracting these patterns as algebraic structures – data dependency algebras (DDAs). This makes it possible to investigate the dependencies of programs independently of any program, e.g., showing that dependency patterns form a category. Further, the execution resources of a parallel computer, its processors and communication network, may be described by a special form of DDAs, called space-time algebras (STAs). The task of programming parallel machines may then be abstracted to embedding a program's DDA into the machine's STA, a task which can be investigated independently of actual programs and physical machines. A basic theory for this was presented in [7], and some practical results were shown in [8].

Although this kind of embedding is central to the parallelization of programs (see, e.g., [9]), very few approaches have tried to abstract the dependency pattern as a separate entity. The theoretical model of synchronous concurrent algorithms [10] seems to be the closest. The functional language Crystal [11] focused very clearly on data dependencies, but these were not made into explicit language elements.

In this paper we utilize our notion of DDAs to study various ways of combining simple DDAs to more complex ones. The constructions are shown to give the category of DDAs the property of having *all limits*. Theorems about the commutativity of limits allow us to factorize complex data dependency and space-time patterns, find embeddings between appropriate factors, and then automatically combine these into an embedding of the program onto the target hardware's space-time.

The paper is organized as follows. In Section 2 we define some mathematical preliminaries. In Section 3 we define the basic concepts of DDAs and embeddings. In Section 4 we show that DDAs and their embeddings form a finitely complete category. Section 5 gives a factorization theorem for embeddings, before we conclude.

2. MATHEMATICAL PRELIMINARIES

The symbol $\mathbb{N} = \{0, 1, ...\}$ denotes the set of natural numbers, and for $n \in \mathbb{N}$, $[n] = \{0, 1, 2, ..., n-1\}$. Thus $[0] = \{\} = \emptyset$ denotes the empty set.

A sequence $\vec{x}$ with length n over X is an array $\vec{x} \in X^{[n]}$. The unique empty sequence is denoted $\lambda \in X^{[0]}$. Concatenation of sequences $\vec{x}$ and $\vec{y}$ is given by juxtaposition $\vec{x}\vec{y}$. For finite $\vec{x} \in X^{[n]}$ and finite $\vec{y} \in X^{[m]}$ we have $\vec{x}\vec{y} \in X^{[n]}X^{[m]} = X^{[n+m]}$ such that, for $i \in [n+m]$,

$$(\vec{x}\vec{y})[i] = \begin{cases} \vec{x}[i] & \text{if } i < n \\ \vec{y}[i-n] & otherwise. \end{cases}$$

The set $X^* = \bigcup_{n \in \mathbb{N}} X^{[n]}$ is the set of all finite sequences from X, and $X^+ = X^* \setminus X^{[0]}$ is the set of all nonempty, finite sequences of elements from X.

The set of sequences with elements from X and length up to n is given by $X^{\lfloor n \rfloor} = \bigcup_{i \in [n+1]} X^{[i]}$, while $X^{\lceil n \rceil} = X^{\lfloor n \rfloor} \setminus X^{[0]}$ are the nonempty sequences with length up to n. Operations on sequences include $x :: \vec{y} \in X^+$, which prepends an element $x \in X$ to a sequence $\vec{y} \in X^*$, i.e.,

$$(x :: \vec{y})[i] = \begin{cases} x & \text{if } i = 0 \\ \vec{y}[i-1] & otherwise \end{cases}$$

and $\vec{y} :: x \in X^+$, which appends an element $x \in X$ to a sequence $\vec{y} \in X^*$.

A *category* **C** consists of a collection $Obj(\mathbf{C})$ of entities called objects, a collection $Mor(\mathbf{C})$ of entities called morphisms, two operations assigning to each morphism f its domain $dom(f)$, which is an object of **C**, and its codomain $cod(f)$, also an object of **C**. Morphisms f and g are composable if $cod(f) = dom(g)$, and the composition is a morphism denoted by $g \circ f$, with $dom(g \circ f) = dom(f)$ and $cod(g \circ f) = cod(g)$. The composition of morphisms is associative. There is also an operation assigning to each object A of **C** an identity morphism $id_A : A \to A$, which is unitary with respect to the composition.

A *functor* F between categories **S** and **C**, written $F : \mathbf{S} \to \mathbf{C}$, is specified by an operation assigning to objects A in **S**, objects FA in **C** and an operation assigning to morphisms $f : A \to B$ in **S**, morphisms $Ff : FA \to FB$ in **C** such that $F(id_A) = id_{FA}$, and whenever the composition of morphisms $g \circ f$ is defined in **S**, we have $F(g \circ f) = Fg \circ Ff$. For every object X of **C** there is a functor $C_X : \mathbf{S} \to \mathbf{C}$ such that $C_X A = X$ and $C_X(f : A \to B) = id_X$, for all $A, B \in Obj(\mathbf{S})$ and $f \in Mor(\mathbf{S})$. A *diagram* of shape **S** in a category **C** is a functor $D : \mathbf{S} \to \mathbf{C}$.

Let **S** and **C** be categories and $F, G : \mathbf{S} \to \mathbf{C}$ be functors. Then a *natural transformation* η from F to G, written $\eta : F \Rightarrow G$, is given by an operation which assigns to each object $A \in \mathbf{S}$ a morphism $\eta_A : FA \to GA$ in **C** such that, for any morphism $f : A \to B$ in **S**, we have $Gf \circ \eta_A = \eta_B \circ Ff$.

A *cone* X for a diagram $D : \mathbf{S} \to \mathbf{C}$ can be seen as a natural transformation $\eta : C_X \Rightarrow D$, where $C_X : \mathbf{S} \to \mathbf{C}$ is the constant functor for the object X in **C**.

A *limit* for a diagram $D : \mathbf{S} \to \mathbf{C}$ is a cone X given by a natural transformation $\eta : C_X \Rightarrow D$ such that if Y is another cone for D given by a natural transformation $\eta' : C_Y \Rightarrow D$, then there exists a unique morphism $k : Y \to X$, called mediator

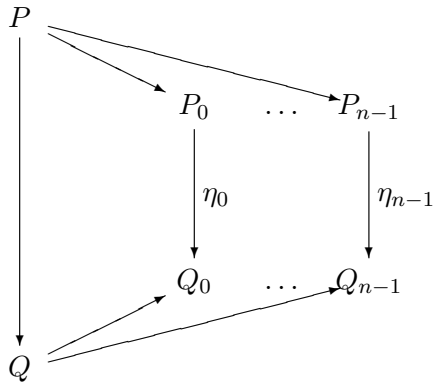
morphism such that $\eta_A \circ k = \eta'_A$, for all $A \in Obj(\mathbf{S})$. Given that **S** has objects $A_0, ..., A_{n-1}$, we denote a diagram D of shape **S** in **C** by $[D_0, ..., D_{n-1}]^{\mathbf{S}}$, where the objects A_i are mapped into the objects D_i; the limit of such a diagram will be denoted by $L([D_0, ..., D_{n-1}]^{\mathbf{S}})$. Some special limits in a category are: a *terminal* object is $L([]^{\mathbf{S}})$, a limit for a diagram of the empty category **S**. The *product* of two objects D_0 and D_1 is $L([D_0, D_1]^{\mathbf{S}})$, where **S** is the category of two objects with only identity morphisms. The *equalizer* of a pair of morphisms $f, g : D_0 \to D_1$ is $L([D_0, D_1])^{\mathbf{S}}$, where **S** is a category

$$A_0\bullet \rightrightarrows \bullet A_1$$

A category is *finitely complete* if for each finite diagram in the category there exists a limit.

Theorem 2.1 (Finite completeness). *A category* **C** *is finitely complete iff it has binary products, equalizers, and terminal objects.*

Theorem 2.2 (Factorization). *Given a category* **C** *and two limit objects* $P \cong L([P_0, ..., P_{n-1}]^{\mathbf{S}})$ *respectively* $Q \cong L([Q_0, ..., Q_{n-1}]^{\mathbf{S}})$ *such that there exists a natural transformation* $\eta : [P_0, ..., P_{n-1}]^{\mathbf{S}} \Rightarrow [Q_0, ..., Q_{n-1}]^{\mathbf{S}}$*, there exists a unique morphism from* P *into* Q *making the diagram*



commute.

Proposition 2.3 (Distributivity of limits). *Given a finitely complete category* **C**, *shapes* **S** *and* $\mathbf{S}'$ *with* n *respectively* n' *nodes, and diagrams* $\boldsymbol{D}^i = [{P_0}^i, ..., {P_{n-1}}^i]^{\boldsymbol{S}}$, $i \in [n']$, *we have that* $L'([..., L(\boldsymbol{D}^i), ...]^{\boldsymbol{S}'}) \cong L([..., L'(\boldsymbol{D}'_j), ...]^{\boldsymbol{S}})$, *where* $\boldsymbol{D}'_j = [{P_j}^0, ..., {P_j}^{n'-1}]^{\boldsymbol{S}'}$, *for* $j \in [n]$.

3. DATA DEPENDENCY ALGEBRAS

3.1. DDAs and embeddings

Definition 3.1. *A DDA* $D = \mathcal{D}\langle P, B, r, s\rangle$ *is given by the data of the 4-tuple such that*

- *P is a set of points,*
- *B is a set of branch indices,*
- *r is the data request consisting of*
  - *the request-guard relation $r_g \subseteq P \times B$ telling which request branches $b \in B$ exist for a point $p \in P$,*
  - *the request-target function $r_t : r_g \to P$ telling which point $r_t(p, b)$ a request goes to, and*
  - *the request-branchback function $r_b : r_g \to B$ telling which supply branch $r_b(p, b)$ leads back from $r_t(p, b)$ to p,*

and

- *s is the data supply consisting of*
  - *the supply-guard relation $s_g \subseteq P \times B$ telling which supply branches $b \in B$ exist for a point $p \in P$,*
  - *the supply-target function $s_t : s_g \to P$ telling which point $s_t(p, b)$ a request goes to, and*
  - *the supply-branchback function $s_b : s_g \to B$ telling which request branch $r_b(p, b)$ leads back from $s_t(p, b)$ to p,*

such that

$$\begin{aligned} r_g(p,b) &\Rightarrow s_g(r_t(p,b), r_b(p,b)), \\ r_g(p,b) &\Rightarrow s_t(r_t(p,b), r_b(p,b)) = p, \\ r_g(p,b) &\Rightarrow s_b(r_t(p,b), r_b(p,b)) = b, \end{aligned}$$

and

$$\begin{aligned} s_g(p,b) &\Rightarrow r_g(s_t(p,b), s_b(p,b)), \\ s_g(p,b) &\Rightarrow r_t(s_t(p,b), s_b(p,b)) = p, \\ s_g(p,b) &\Rightarrow r_b(s_t(p,b), s_b(p,b)) = b. \end{aligned}$$

Example 3.2. The loop DDA with points P is defined by $\boldsymbol{L}(P) = \mathcal{D}\langle P, B, c, c\rangle$ with $B = [1]$, $c_g = P \times B$, $c_t(p, b) = b$.

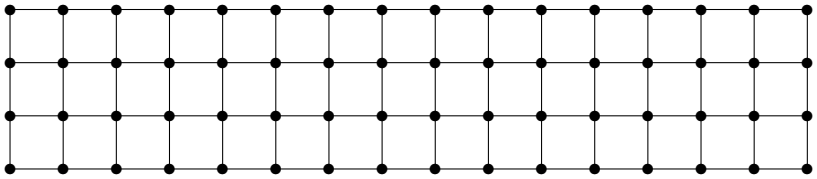
The loop DDA $\boldsymbol{L}([13])$:

Example 3.3. A linear time DDA over $P \subseteq \mathbb{N}$ is $\boldsymbol{T}(1, P) = \mathcal{D}\langle P, [1], t, u\rangle$ with $t_g = \{\langle p, b\rangle \in P \times [1] \mid p + 1 \in P\}$ as the request-guard, $t_t(p, b) = p + 1$ as the request-target, $t_b(p, b) = b$ as the request-branchback, $u_g = \{\langle p, b\rangle \in P \times [d] \mid p - 1 \in P\}$ as the supply-guard, $u_t(p, b) = p - 1$ as the supply-target, and $u_b(p, b) = b$ as the supply-branchback.

The time DDA $\boldsymbol{T}(1, 4)$:

Example 3.4. A *k*-dimensional grid with radius $r \in \mathbb{N}$ for $\vec{g} = \langle g_0, ..., g_{k-1} \rangle \in \mathbb{N}^{[k]}$ is $\boldsymbol{G}(r, \vec{g}) = \mathcal{D}\langle G, B, f, f \rangle$ with $G = [g_0] \times ... \times [g_{k-1}]$ as the points (g_i is the number of points in dimension $i+1$), $B = [k] \times \{-r, ..., -1, 1, ..., r\}$ as the branch indices, $f_g(\langle p_0, ..., p_i, ..., p_{k-1} \rangle, \langle i, j \rangle) = 0 \leq p_i + j < g_i$ as the request-guard and supply-guard, $f_t(\langle p_0, ..., p_i, ..., p_{k-1} \rangle, \langle i, j \rangle) = \langle p_0, ..., p_i + j, ..., p_{k-1} \rangle$ as the request-target and supply-target, $f_b(p, \langle i, j \rangle) = \langle i, -j \rangle$ as the request-branchback and supply-branchback.

The two-dimensional grid $\boldsymbol{G}(1, \langle 4, 16 \rangle)$:

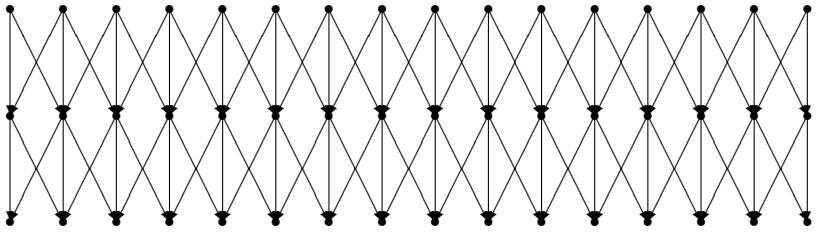


Example 3.5. A *k*-dimensional nearest-neighbour DDA with radius $r \in \mathbb{N}$ and steps $T \subseteq \mathbb{N}$ is $\boldsymbol{N}(r, \vec{n}, T) = \mathcal{D}\langle N, B, p, n \rangle$, where $\vec{n} = \langle n_0, ..., n_{k-1} \rangle \in \mathbb{N}^{[k]}$ (n_i is the number of points in dimension $i+1$), such that $N = ([n_0] \times ... \times [n_{k-1}]) \times T$, $B = [k] \times \{-r, ..., -1, 1, ..., r\}$,

$$
\begin{aligned}
p_g(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= (0 \leq p_i + j < n_i) \wedge (t - 1 \in T), \\
p_t(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= \langle \langle p_0, ..., p_i + j, ..., p_{k-1} \rangle, t - 1 \rangle, \\
p_b(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= \langle i, -j \rangle, \\
n_g(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= (0 \leq p_i + j < n_i) \wedge (t + 1 \in T), \\
n_t(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= \langle \langle p_0, ..., p_i + j, ..., p_{k-1} \rangle, t + 1 \rangle, \\
n_b(\langle \langle p_0, ..., p_i, ..., p_{k-1} \rangle, t \rangle, \langle i, j \rangle) &= \langle i, -j \rangle.
\end{aligned}
$$

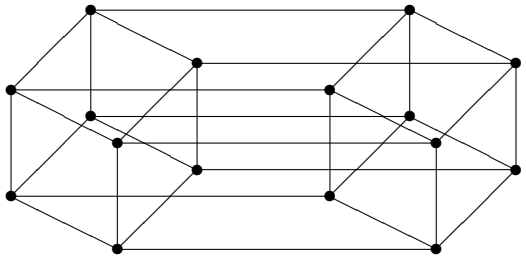
The points are indexed by a grid reference $\langle p_1, ..., p_k \rangle$ and by row number t. The branch index $\langle i, j \rangle$ tells which dimension, $i + 1$, the movement is along, and how far, j, the movement is.

The nearest-neighbour DDA $\boldsymbol{N}(1, \langle 16 \rangle, [3])$:



Example 3.6. A hypercube of dimension $m \in \mathbb{N}$ is given by $\mathbf{H}(m) = \mathcal{D}\langle H, B, h, h\rangle$ with $H = [2]^{[m]}$ as the points, $B = [m]$ as the branch indices, $h_g = H \times B$ as the request-guard and supply-guard, $h_t(\langle p_0, ..., p_b, ..., p_{m-1}\rangle, b) = \langle p_0, ..., 1 - p_b, ..., p_{m-1}\rangle$ as the request-target and supply-target, and $h_b(\vec{p}, b) = b$ as the request-branchback and supply-branchback.

The hypercube connections drawn for $\boldsymbol{H}(4)$:



Proposition 3.7. *Given a DDA $D = \mathcal{D}\langle P, B, r, s\rangle$ and a set $C \subseteq B$ such that for all $\langle p, b\rangle \in r_g$ when $b \in C$, we have $r_b(p, b) \in C$, and for all $\langle p, b\rangle \in s_g$ when $b \in C$, we have $s_b(p, b) \in C$. Then ${}^C D = \mathcal{D}\langle P, C, {}^C r, {}^C s\rangle$ is a DDA, where ${}^C r_g = r_g \cap (P \times C)$ and ${}^C s_g = s_g \cap (P \times C)$, and for all $\langle p, b\rangle \in {}^C r_g$, ${}^C r_t(p, b) = r_t(p, b)$ and ${}^C r_b(p, b) = r_b(p, b)$, and for all $\langle p, b\rangle \in {}^C s_g$, ${}^C s_t(p, b) = s_t(p, b)$ and ${}^C s_b(p, b) = s_b(p, b)$.*

As in the case of graphs where sequences of edges give us paths, we get paths in a DDA from sequences of branch indices.

Definition 3.8. *Given a DDA $\mathcal{D}\langle P, B, r, s\rangle$ and a sequence $\vec{c} \in B^*$. Then $\vec{c}$ is a request-path from $p \in P$ if $\vec{c}$ is the empty sequence $\lambda \in B^{[0]}$, or $\vec{c} = b :: \vec{d}$ such that $r_g(p, b)$ and $\vec{d} \in B^*$ is a request-path from $r_t(p, b)$. Likewise, $\vec{c}$ is a supply-path from $p \in P$ if $\vec{c}$ is the empty sequence λ, or $\vec{c} = b :: \vec{d}$ such that $s_g(p, b)$ and $\vec{d} \in B^*$ is a supply-path from $s_t(p, b)$.*

Now the request and supply operators can be extended to encompass paths.

Definition 3.9. *Given a DDA $\mathcal{D}\langle P, B, r, s\rangle$. Define a path data request $\overline{r}$ and a path data supply $\overline{s}$ with $\overline{r}_g$, $\overline{s}_g \subseteq P \times B^*$, $\overline{r}_t : \overline{r}_g \to P$, $\overline{r}_b : \overline{r}_g \to B^*$, $\overline{s}_t : \overline{s}_g \to P$, $\overline{s}_b : \overline{s}_g \to B^*$ by setting*

- *for the empty sequence $\lambda \in B^{[0]}$,*

$$\begin{array}{rcll} \langle p, \lambda\rangle & \in & \overline{r}_g, & \langle p, \lambda\rangle \in \overline{s}_g, \\ \overline{r}_t(p, \lambda) & = & p, & \overline{s}_t(p, \lambda) = p, \\ \overline{r}_b(p, \lambda) & = & \lambda, & \overline{s}_b(p, \lambda) = \lambda, \end{array}$$

- *for any nonempty sequence $\vec{c} = b :: \vec{d} \in B^+$,*

$$
\begin{aligned}
\overline{r}_g(p,\vec{c}) &= r_g(p,b) \wedge \overline{r}_g(r_t(p,b),\vec{d}),\\
\overline{r}_t(p,\vec{c}) &= \overline{r}_t(r_t(p,b),\vec{d}),\\
\overline{r}_b(p,\vec{c}) &= \overline{r}_b(r_t(p,b),\vec{d}) :: r_b(p,b),\\
\overline{s}_g(p,\vec{c}) &= s_g(p,b) \wedge \overline{s}_g(s_t(p,b),\vec{d}),\\
\overline{s}_t(p,\vec{c}) &= \overline{s}_t(s_t(p,b),\vec{d}),\\
\overline{s}_b(p,\vec{c}) &= \overline{s}_b(s_t(p,b),\vec{d}) :: s_b(p,b).
\end{aligned}
$$

Proposition 3.10. *Given a DDA $D = \mathcal{D}\langle P, B, r, s\rangle$. For all natural numbers $n \in \mathbb{N}$, $D^{[n]} = \mathcal{D}\langle P, B^{[n]}, {}^{(B^{[n]})}\overline{r}, {}^{(B^{[n]})}\overline{s}\rangle$ will be a DDA, where ${}^{(B^{[n]})}\overline{r}_g = \overline{r}_g \cap (P \times B^{[n]})$ and ${}^{(B^{[n]})}\overline{s}_g = \overline{s}_g \cap (P \times B^{[n]})$.*

Corollary 3.11. *Given a DDA $D = \mathcal{D}\langle P, B, r, s\rangle$ and $n \in \mathbb{N}$. For $\bullet = [n], \lceil n\rceil, \lfloor n\rfloor, +, *$ we have that $D^\bullet = \mathcal{D}\langle P, B^\bullet, {}^{(B^\bullet)}\overline{r}, {}^{(B^\bullet)}\overline{s}\rangle$ is a DDA.*

For any DDA D, the construction $D^{\lfloor 1\rfloor}$ adds a loop at every node.

Definition 3.12. *Given DDAs $D = \mathcal{D}\langle P, B, r, s\rangle$ and $F = \mathcal{D}\langle Q, C, v, w\rangle$ and a nonzero number $n \in \mathbb{N}, n > 0$. For $\bullet = [n], \lceil n\rceil, +$, a $\bullet$-embedding $e : D \to F$ is given by a collection of functions*

- *$e_p : P \to Q$ mapping the points,*
- *$e_r : r_g \to C^\bullet$ mapping request-targets to request-paths, and*
- *$e_s : s_g \to C^\bullet$ mapping supply-targets to supply-paths,*

such that

$$
\begin{aligned}
r_g(p,b) &\Rightarrow {}^{(C^\bullet)}\overline{v}_g(e_p(p), e_r(p,b)),\\
r_g(p,b) &\Rightarrow e_p(r_t(p,b)) = {}^{(C^\bullet)}\overline{v}_t(e_p(p), e_r(p,b)),\\
r_g(p,b) &\Rightarrow e_s(r_t(p,b), r_b(p,b)) = {}^{(C^\bullet)}\overline{v}_b(e_p(p), e_r(p,b)),\\
s_g(p,b) &\Rightarrow {}^{(C^\bullet)}\overline{w}_g(e_p(p), e_s(p,b)),\\
s_g(p,b) &\Rightarrow e_p(s_t(p,b)) = {}^{(C^\bullet)}\overline{w}_t(e_p(p), e_s(p,b)),\\
s_g(p,b) &\Rightarrow e_r(s_t(p,b), s_b(p,b)) = {}^{(C^\bullet)}\overline{w}_b(e_p(p), e_s(p,b)).
\end{aligned}
$$

If we have an embedding $e : D \to F$, we get an embedding $e^{\lfloor 1\rfloor} : D^{\lfloor 1\rfloor} \to F^{\lfloor 1\rfloor}$, which is the same as e and maps the added loops in $D^{\lfloor 1\rfloor}$ to the corresponding added loops in $F^{\lfloor 1\rfloor}$.

Proposition 3.13. *Given DDAs $D = \mathcal{D}\langle P, B, r, s\rangle$ and $F = \mathcal{D}\langle Q, C, v, w\rangle$ and $n \in \mathbb{N}$. For $\bullet = [n+1], \lceil n\rceil, +$, an embedding $e : D \to F$ extends elementwise to an embedding $\overline{e} : D^\bullet \to F$.*

Proof. Define $\overline{e}$ by $\overline{e}_p = e_p(p)$, and, if $\bullet = \lceil 0\rceil$, then $\overline{e}_r$ and $\overline{e}_s$ are the empty functions, otherwise

$$\overline{e}_r(p, b :: \vec{c}) = \begin{cases} e_r(p, b) & \text{if } \vec{c} = \lambda, \\ e_r(p, b)\overline{e}_r(r_t(p, b), \vec{c}) & \text{if } \vec{c} \neq \lambda, \end{cases}$$

$$\overline{e}_s(p, b :: \vec{c}) = \begin{cases} e_s(p, b) & \text{if } \vec{c} = \lambda, \\ e_s(p, b)\overline{e}_s(r_t(p, b), \vec{c}) & \text{if } \vec{c} \neq \lambda. \end{cases}$$

It is easy to see that $\overline{e} = \langle \overline{e}_p, \overline{e}_r, \overline{e}_s \rangle$ is indeed an embedding. □

Proposition 3.14. *Given embeddings $e : D \to D'$ and $e' : D' \to D''$. Define*

$$\begin{aligned} (e' \circ e)_p(p) &= \overline{e'}_p(e_p(p)), \\ (e' \circ e)_r(\langle p, b \rangle) &= \overline{e'}_r(\langle e_p(p), e_r(\langle p, b \rangle) \rangle), \\ (e' \circ e)_s(\langle p, b \rangle) &= \overline{e'}_s(\langle e_p(p), e_s(\langle p, b \rangle) \rangle). \end{aligned}$$

The triple $\langle (e' \circ e)_p, (e' \circ e)_r, (e' \circ e)_s \rangle$ yields an embedding from D into D'' denoted $e' \circ e$ and called the composition of e' and e.

Proof. The embedding e defines paths in D' for every branch in D. The embedding $\overline{e'}$ maps paths in D' to paths in D'', specifically it will map the subset of paths given by e. Moreover, this subset consists of matching request and supply branches, so that the requirements on an embedding are satisfied by the composition $e' \circ e$. □

Fact 3.15. Composition of embeddings is associative.

Proposition 3.16. *For any DDA $D = \mathcal{D}\langle P, B, r, s \rangle$ define, $id_p : P \to P$, $id_r : r_g \to B$, and $id_s : s_g \to B$ by*

$$\begin{aligned} id_p(p) &= p, \\ id_r(p, b) &= b, \\ id_s(p, b') &= b' \end{aligned}$$

for all $p \in P$, $\langle p, b \rangle \in r_g$ and $\langle p, b' \rangle \in s_g$. The triple $\langle id_p, id_r, id_s \rangle$ yields an embedding denoted by id_D and called the identity embedding on D.

Proof. From the definitions above it follows that id_D is indeed an embedding from D into D. Let $f : D \to F$ be an arbitrary embedding with $F = \mathcal{D}\langle Q, C, \upsilon, w \rangle$. We have $(f \circ id_D)_p(p) = f_p(id_p(p)) = f_p(p)$ and $(f \circ id_D)_r(p, b) = f_r(id_p(p), id_r(p, b)) = f_r(p, b)$, for all $p \in P$ and $\langle p, b \rangle \in r_g$. Likewise we get that $(f \circ id_D)_s(p, b) = f_s(p, b)$, for all $\langle p, b \rangle \in s_g$. Thus $f \circ id_D = f$. Similarly, for every embedding $g : C \to D$ it can be shown that $id_D \circ g = id_D$. Hence, id_D satisfies the identity axioms. □

3.2. Space-time algebras

Definition 3.17. *A space-time algebra is a data dependency algebra with the following restrictions*:

• *the carrier for the nodes of a space-time algebra is (a subset of) the Cartesian product of the processing elements of a parallel machine and a time counter, usually the integers,*

• *the carrier for the directions are channels going out from and leading into the processors, including a channel for in-memory communication, allowing a processor to retain data in memory between computations, and*

• *the r and s functions define allowable communications from one time-step to the next.*

4. CATEGORY OF DATA DEPENDENCY ALGEBRAS

Definition 4.1. *The category $\mathcal{DDA}$ of DDAs and embeddings consists of*

• *DDAs as objects,*

• *embeddings as morphisms,*

• *the composition operation as in Proposition* 3.14*,*

• *the identity operation as in Proposition* 3.16*.*

The category $\mathcal{DDA}$ defined above is indeed a category since the composition of any two given embeddings $e : D \to D'$ and $e' : D' \to D''$ is an embedding from D to D'', the composition of embeddings is associative, and for any embedding $e : D \to D'$, the identity embeddings on D and D' satisfy the identity axiom: $id_{D'} \circ e = e$ and $e \circ id_D = e$.

In the following we show that the category $\mathcal{DDA}$ has binary products, equalizers, and terminal objects.

Proposition 4.2. *For any two given objects $E = \mathcal{D}\langle N, A, t, u\rangle$ and $F = \mathcal{D}\langle Q, C, v, w\rangle$ of the category $\mathcal{DDA}$ the object $E \times F = \mathcal{D}\langle P, B, r, s\rangle$ together with the embeddings $\pi_E : E \times F \to E$ and $\pi_F : E \times F \to F$ is the categorical product of the DDAs E and F, where $P = N \times Q$, $B = A \times C$, $r_g \subseteq P \times B$, $r_t : r_g \to P$, $r_b : r_g \to B$, $s_g \subseteq P \times B$, $s_t : r_g \to P$, and $s_b : r_g \to B$ such that*

$$\begin{aligned}
r_g((n,q),(a,c)) &= t_g(n,a) \wedge v_g(q,c),\\
r_t((n,q),(a,c)) &= (t_t(n,a), v_t(q,c)),\\
r_b((n,q),(a,c)) &= (t_b(n,a), v_b(q,c)),\\
s_g((n,q),(a,c)) &= u_g(n,a) \wedge w_g(q,c),\\
s_t((n,q),(a,c)) &= (u_t(n,a), w_t(q,c)),\\
s_b((n,q),(a,c)) &= (u_b(n,a), w_b(q,c)),
\end{aligned}$$

and $(\pi_E)_p : N \times Q \to N$, $(\pi_E)_r : r_q \to A$, $(\pi_E)_s : s_q \to A$, $(\pi_F)_p : N \times Q \to Q$, $(\pi_F)_r : r_q \to C$, $(\pi_F)_s : s_q \to C$ such that

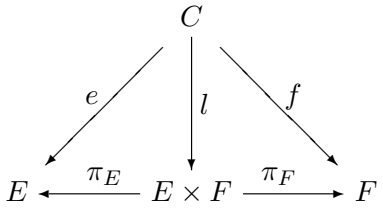
$$
\begin{aligned}
(\pi_E)_p(n,q) &= n,\\
(\pi_E)_r((n,q),(a,c)) &= a,\\
(\pi_E)_s((n,q),(a',c')) &= a',\\
(\pi_F)_p(n,q) &= q,\\
(\pi_F)_r((n,q),(a,c)) &= c,\\
(\pi_F)_s((n,q),(a',c')) &= c',
\end{aligned}
$$

for all $\langle n,q\rangle \in N \times Q$, $\langle a,c\rangle \in r_g$ *and* $\langle a',c'\rangle \in s_g$.

Proof. It can be easily verified that $E \times F$ is indeed a DDA and that π_E and π_F satisfy the axioms for an embedding. Let $C = \mathcal{D}\langle M, D, x, y\rangle$ be a $\mathcal{DDA}$ object and consider two embeddings $e : C \to E$, respectively $f : C \to F$. We define the following mappings $l_p : M \to P$, $l_r : x_g \to A^* \times C^*$, and $l_s : y_g \to A^* \times C^*$ such that

$$
\begin{aligned}
l_p(m) &= (e_p(m), f_p(m)),\\
l_r(m,d) &= (e_r(m,d), f_r(m,d)),\\
l_s(m,d') &= (e_s(m,d'), f_s(m,d')),
\end{aligned}
$$

for all $m \in M$, $\langle m,d\rangle \in x_g$, and $\langle m,d'\rangle \in y_g$. Since e and f are embeddings, it can be easily verified that $l = \langle l_p, l_r, l_s\rangle$ is an embedding from C into $E \times F$. In the following we show that l makes the diagram below commute and it is unique with respect to that property.



We have

$(\pi_E \circ l)_p(m) = (\pi_E)_p(l_p(m)) = (\pi_E)_p(e_p(m), f_p(m)) = e_p(m)$,
$(\pi_E \circ l)_r(m,d) = (\overline{\pi}_E)_r(l_p(m), l_r(m,d)) = e_r(m,d)$,
$(\pi_E \circ l)_s(m,d') = (\overline{\pi}_E)_s(l_p(m), l_s(m,d')) = e_s(m,d')$,

and

$(\pi_F \circ l)_p(m) = (\pi_F)_p(l_p(m)) = (\pi_F)_p(e_p(m), f_p(m)) = f_p(m)$,
$(\pi_F \circ l)_r(m,d) = (\overline{\pi}_F)_r(l_p(m), l_r(m,d)) = f_r(m,d)$,
$(\pi_F \circ l)_s(m,d') = (\overline{\pi}_F)_s(l_p(m), l_s(m,d')) = f_s(m,d')$,

for all $m \in M$, $\langle m,d\rangle \in x_g$, and $\langle m,d'\rangle \in y_g$.

To verify the uniqueness of l, we assume that there exists an embedding $l' : C \to E \times F$ with

$$\begin{aligned} l'_p(m) &= (a_p(m), b_p(m)), \\ l'_r(m,d) &= (a_r(m,d), b_r(m,d)), \\ l'_s(m,d') &= (a_s(m,d'), b_s(m,d')), \end{aligned}$$

where $m \in M$, $\langle m,d\rangle \in x_g$, $\langle m,d'\rangle \in y_g$, and $a_p : M \to N$, $b_p : M \to Q$, $a_r : x_g \to A^*$, $b_r : x_g \to C^*$, $a_s : y_g \to A^*$, $b_s : y_g \to C^*$ are mappings such that

$$\begin{aligned} (\pi_E \circ l')_p(m) &= e_p(m), \\ (\pi_E \circ l')_r(m,d) &= e_r(m,d), \\ (\pi_E \circ l')_s(m,d') &= e_s(m,d'), \\ (\pi_F \circ l')_p(m) &= f_p(m), \\ (\pi_F \circ l')_r(m,d) &= f_r(m,d), \\ (\pi_F \circ l')_s(m,d') &= f_s(m,d'), \end{aligned}$$

for all $m \in M$, $\langle m,d\rangle \in x_g$, and $\langle m,d'\rangle \in y_g$.

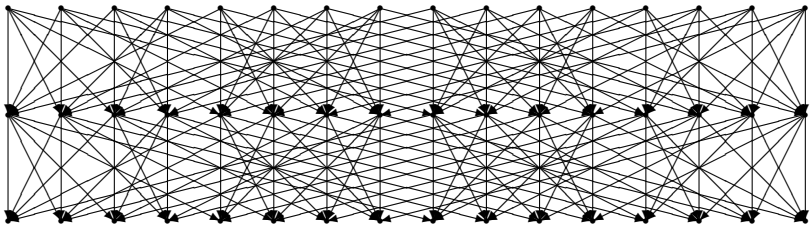
Applying now the definitions of p_1, p_2, l', we obtain that

$$\begin{aligned} a_p(m) &= e_p(m), \\ a_r(m,d) &= e_r(m,d), \\ a_s(m,d') &= e_s(m,d'), \\ b_p(m) &= f_p(m), \\ b_r(m,d) &= f_r(m,d), \\ b_s(m,d') &= f_s(m,d'), \end{aligned}$$

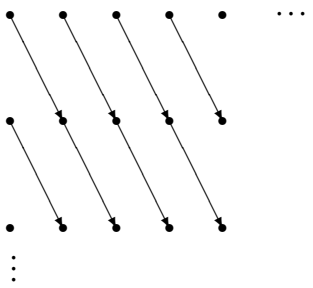
for all $m \in M$, $\langle m,d\rangle \in x_g$, and $\langle m,d'\rangle \in y_g$, thus $l = l'$. □

The space-time algebra of a given DDA *A* is the product of *A* with a time DDA. Typically we will add selfloops to a hardware communication network when creating its space-time. This is because loops represent data in memory between time-steps.

Example 4.3. The hypercube space-time pattern $\boldsymbol{H}^{\lfloor 1 \rfloor}(4) \times \boldsymbol{T}(1,(3))$:



Example 4.4. A finite product of linear time DDAs yields a DDA as depicted below:



Example 4.5. The nearest-neighbour DDA is isomorphic to the reflexive grid DDA in time, i.e., $\boldsymbol{N}(r, \vec{g}, P) \cong \boldsymbol{G}^{\lfloor 1 \rfloor}(r, \vec{g}) \times \boldsymbol{T}(1, P)$. Example 3.5 illustrates the composite construction $\boldsymbol{N}(1, \langle 16 \rangle, [3]) \cong \boldsymbol{G}^{\lfloor 1 \rfloor}(1, \langle 16 \rangle) \times \boldsymbol{T}(1, [3])$.

Proposition 4.6. *For any given two DDAs $F = \mathcal{D}\langle P, B, r, s\rangle$, $G = \mathcal{D}\langle M, D, x, y\rangle$ and embeddings $f : F \to G$, $g : F \to G$ of the category $\mathcal{DDA}$ the embedding $e : E \to F$ is the equalizer of the embeddings f and g, where $E = \mathcal{D}\langle N, A, t, u\rangle$ with*

- $N = \{p \in P : f_p(p) = g_p(p)\}$;
- $A = \{a \in B^* : (\exists p \in N : (\overline{f}(p, a) = \overline{g}(p, a)) \wedge (\forall a', a'' \in B^* \setminus \{\lambda\} : (a' :: a'' = a) \wedge \neg(\overline{f}(p, a') = \overline{g}(p, a') \wedge \overline{f}(p, a'') = \overline{g}(p, a'))))\}$;
- *the data request t consisting of*
  - $t_g = \{\langle p, a\rangle \in \overline{r}_g : p \in N \wedge a \in A\}$,
  - $t_t : t_g \to N$ s.t. $t_t(p, a) = \overline{r}_t(p, a)$ *for all* $\langle p, a\rangle \in t_g$,
  - $t_b : t_g \to A$ s.t. $t_b(p, a) = \overline{r}_b(p, a)$ *for all* $\langle p, a\rangle \in t_g$;
- *the data supply u consisting of*
  - $u_g = \{\langle p, a\rangle \in \overline{s}_g : p \in N \wedge a \in A\}$,
  - $u_t : u_g \to N$ *such that* $u_t(p, a) = \overline{s}_t(p, a)$ *for all* $\langle p, a\rangle \in u_g$,
  - $u_b : u_g \to A$ *such that* $u_b(p, a) = \overline{s}_b(p, a)$ *for all* $\langle p, a\rangle \in u_g$,

and e given by the triple $\langle e_p, e_r, e_s\rangle$ with

$$\begin{aligned} e_p(p) &= p, \\ e_r(p, b) &= b, \\ e_s(p', b') &= b', \end{aligned}$$

for all $p \in N$, $\langle p, b\rangle \in t_g$, and $\langle p', b'\rangle \in u_g$.

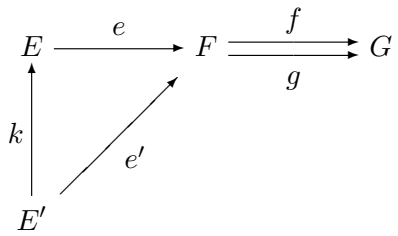
Proof. A closer examination of the definitions above yields that $E = \mathcal{D}\langle N, A, t, u\rangle$ is indeed a DDA and $e : E \to F$ satisfies the axioms for an embedding. In the following we show that e is indeed the equalizer of the embeddings f and g. We have

$$
\begin{aligned}
&(f \circ e)_p(p) = f_p(e_p(p)) = f_p(p) = g_p(p) = g_p(e_p(p)) = (g \circ e)_p(p),\\
&(f \circ e)_r(p,a) = \overline{f}_r(e_p(p), e_r(p,a)) = \overline{f}_r(p,a) = \overline{g}_r(p,a) = \overline{g}(e_p(p), e_r(p,a))\\
&\qquad = (g \circ e)_r(p,a),\\
&(f \circ e)_s(p,a) = \overline{f}_s(e_p(p), e_s(p,a)) = \overline{f}_s(p,a) = \overline{g}_s(p,a) = \overline{g}(e_p(p), e_s(p,a))\\
&\qquad = (g \circ e)_s(p,a),
\end{aligned}
$$

for all $p \in N$, $\langle p, b\rangle \in t_g$ and $\langle p', b'\rangle \in u_g$.

Suppose that there is another embedding $e' : E' \to F$ such that $f \circ e' = g \circ e'$. Let $E' = \mathcal{D}\langle Q, C, v, w\rangle$. We have that $e'_p(q) \in N$ for all $q \in Q$ since $(f \circ e')_p(q) = (g \circ e')_p(q)$ for all $q \in Q$. Let $\langle q, c\rangle$ be an arbitrary request-guard pair from v_g. Since $(f \circ e')_r(q,c) = (g \circ e')_r(q,c)$, we get that $e'_r(q,c) \in A$ or $e'_r(q,c) = a_0 :: a_2 ... :: a_{k-1}$ with $a_i \in A \setminus \{\lambda\}, i \in [k]$. We can reason similarly for any supply-guard pair w_g.

Consider now the diagram



where $k : E' \to E$ is given by the triple $\langle k_p, k_r, k_s\rangle$ with

$$
\begin{aligned}
k_p(q) &= e'_p(q),\\
k_r(q,c) &= e'_r(q,c),\\
k_s(q',c') &= e'_s(q',c'),
\end{aligned}
$$

for all $q \in Q$, $\langle q, c\rangle \in v_g$, and $\langle q', c'\rangle \in w_g$. Since $e' : E' \to F$ is an embedding, it can be easily verified that $k = \langle k_p, k_r, k_s\rangle$ is an embedding from E' into E.

In order to show that $e' = e \circ k$, we have to verify that the following equalities hold:

$$
\begin{aligned}
e'_p(q) &= (e \circ k)_p(q), &(1)\\
e'_r(q,c) &= (e \circ k)_r(q,c), &(2)\\
e'_s(q',c') &= (e \circ k)_s(q',c'), &(3)
\end{aligned}
$$

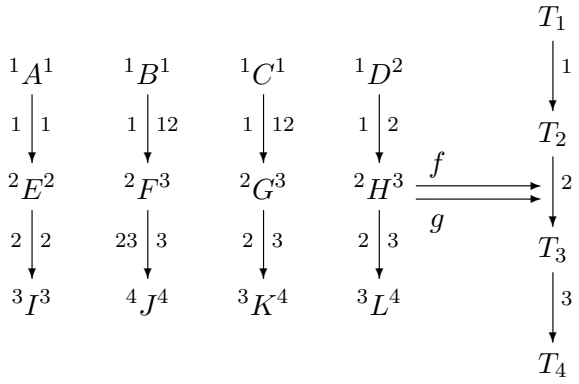
for all $q \in Q$, $\langle q, c\rangle \in v_g$, and $\langle q', c'\rangle \in w_g$.

We will prove only (2), since (3) can be proved similarly and (1) follows directly from applying the definition of k and considering that e_p is an inclusion.

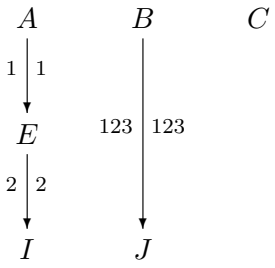
To prove (2), let $\langle q, c\rangle \in v_g$. There are two cases to consider: if $e'_r(q,c) \in A$, we have $(e \circ k)_r(q,c) = e_r(e'_r(q,c)) = e'(q,c)$; if $e'(q,c) = a_0 :: a_1 :: ... :: a_{k-1}$ with $a_i \in A \setminus \{\lambda\}, i \in [k]$, then $(e \circ k)_r(q,c) = \overline{e}_r(k_p(q), a_0 :: a_1 :: ... :: a_{k-1}) = e_r(k_p(q), a_0) :: \overline{e}_r(t_t(k_p(q), a_0), a_1 :: ... :: a_{k-1}) = a_0 :: \overline{e}_r(t_t(k_p(q), a_0), a_1 :: ... :: a_{k-1}) = ... = a_0 :: a_1 :: ... :: a_{k-1} = e'(q,c)$.

Finally, suppose that there exists k' such that $e' = e \circ k'$. Applying the definition of e, we obtain that $k'_p = e'_p$, $k'_r = e'_r$, $k'_s = e'_s$, thus $k = k'$. □

Example 4.7. Given a DDA F with nodes $\{A, ..., L\}$ and a time-DDA $\boldsymbol{T}(1,4)$ with nodes $\{T_1, ..., T_4\}$ and arc indices $\{1,2,3\}$. Consider embeddings $f, g : F \to \boldsymbol{T}(1,4)$



defined as follows: $f_p({}^iX^j) = T_i$ respectively $g_p({}^iX^j) = T_j$, for $X = A, ..., L$ and $i, j = 1, ..., 4$. In the same way, f_r (resp. g_r) maps a request-guard pair from F into a request-arc or request-path in G, tagged with the left (resp. right) tag of the source request-arc. Then the equalizer of the diagram above is an embedding $e : E \to F$ with E shown below and e encoded as either the embeddings f or g (since they will coincide on the data from E).



Notice how the equalizer replaced the path from B to J with an arc.

Proposition 4.8. *The loop DDA $\boldsymbol{L}([1])$ is a terminal object in the category $\mathcal{DDA}$.*

Proof. Given an arbitrary DDA $D = \mathcal{D}\langle P, B, r, s\rangle$, we can construct a unique embedding $e : D \to \boldsymbol{L}([1])$, by mapping the points from P into the single point from the point index set of $\boldsymbol{L}([1])$ and by mapping the request-guard (resp. supply-guard) pairs into the single branch index from the branch index set of $\boldsymbol{L}([1])$. □

4.1. Finite completeness

Proposition 4.9. *The category $\mathcal{DDA}$ is finitely complete.*

Proof. Follows from the completeness Theorem 2.1 and Propositions 4.2, 4.6, and 4.8. □

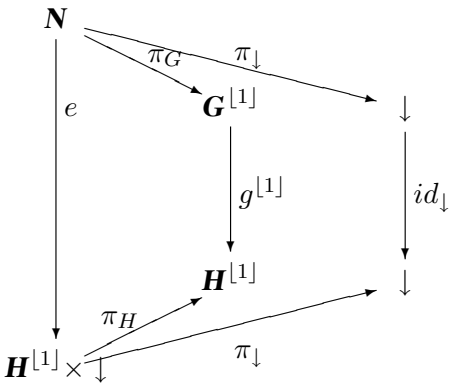
Proposition 4.10. *Given a shape* **S** *with n nodes and linear time DDAs $T_0, ..., T_{n-1}$, there exists an embedding from $L([T_0, ..., T_{n-1}]^{\mathbf{S}})$ into a linear time DDA T.*

Proof. Following the steps from the general limit construction, we get that the limit $L([T_0, ..., T_{n-1}]^{\mathbf{S}})$ is the equalizer E of two embeddings from $F = \prod_{I \in U} T_I$ into $G = \prod_{(I \xrightarrow{e} J \in V)} T_J$, where U is the set of nodes from **S**, and V is the set of edges from **S**. From Example 4.4 we have that F contains points and diagonal arrows, and from the definition of the equalizer we can deduce that E will have a subset of these points connected with diagonal arcs. E will contain the nodes and arrows on which the embeddings f and g are equal, and replace sequences of lost arcs, where f and g give equal paths, with new arcs. We can embed such a DDA into a linear time DDA *T* by embedding each diagonal individually, mapping consecutive nodes of E into consecutive nodes of *T*. □

We will call such an embedding a folding.

5. DDA DECOMPOSITIONS AND PARALLELISM

As pointed out in Example 4.5, the nearest-neighbour DDA $\boldsymbol{N}(1, \langle 16\rangle, [3])$ can be decomposed into the product of the reflexive grid $\boldsymbol{G}^{\lfloor 1 \rfloor}(1, \langle 16\rangle)$ and the time DDA $\boldsymbol{T}(1, [3])$. The embedding of $\boldsymbol{G}(1, \langle 16\rangle)$ into $\boldsymbol{H}(4)$ is typically achieved by using the Grey code embedding (see [7]). We denote this embedding by g. Then we have the diagram



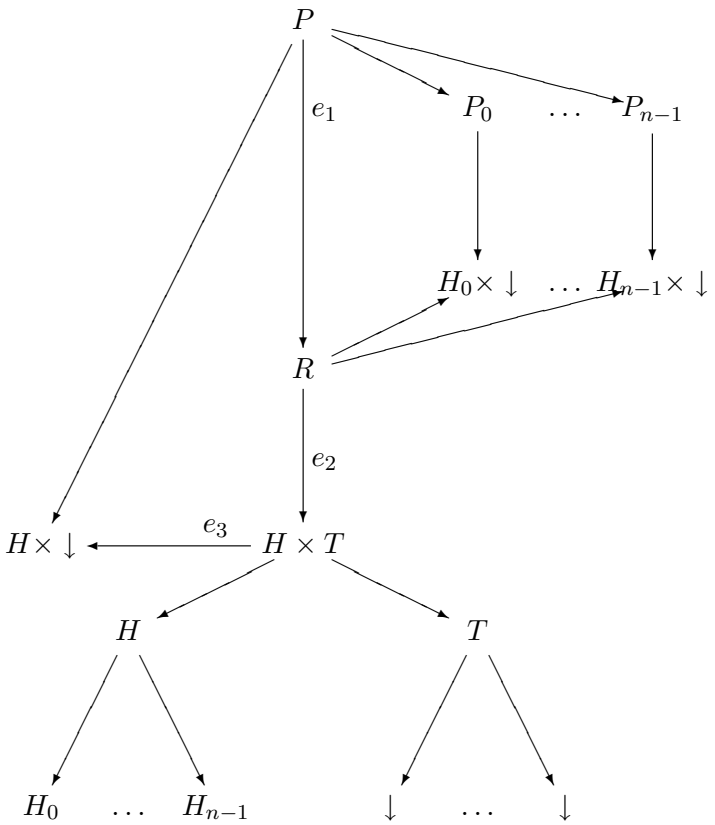
where the arrows $\downarrow$ stand for time the DDA $\boldsymbol{T}(1, [3])$, $id_\downarrow$ is the identity embedding and $\pi_G, \pi_H, \pi_\downarrow$ are projection embeddings. From Theorem 2.2 we get an

embedding e from the nearest-neighbour DDA $\mathbf{N}(1, \langle 16\rangle, [3])$ into the space-time of the hypercube $\boldsymbol{H}^{\lfloor 1\rfloor}(4)$, where e is the mediator from the cone determined by $\mathbf{N}$, $g^{\lfloor 1\rfloor} \circ \pi_G$ and $\pi_\downarrow$ to the limit $\boldsymbol{H}^{\lfloor 1\rfloor} \times \downarrow$.

Now we can look at more complex decomposition diagrams.

Theorem 5.1. *Given a shape* $\mathbf{S}$ *with* n *nodes and DDAs P and H with* $P \cong L([P_0, ..., P_{n-1}]^{\mathbf{S}})$ *and* $H \cong L([H_0, ..., H_{n-1}]^{\mathbf{S}})$ *such that there exists a natural transformation* $\eta : [P_0, ..., P_{n-1}]^{\mathbf{S}} \Rightarrow [H_0 \times \downarrow, ..., H_{n-1} \times \downarrow]^{\mathbf{S}}$*, there exists an embedding from the DDA P into the space-time algebra of H.*

Proof. Consider the diagram



where the arrows $\downarrow$ stand for time DDAs. Since the category $\mathcal{DDA}$ is finitely complete (Proposition 4.9), we can construct the limit $R = L([H_0 \times \downarrow, ..., H_{n-1} \times \downarrow]^{\mathbf{S}})$ and then from the hypothesis and Theorem 2.2 we get that there exists a unique embedding e_1 from *P* into *R*. Furthermore, applying the distributivity property of limits (Proposition 2.3), we obtain an embedding e_2 (isomorphism) from R into $H \times T$, where $T = L([\downarrow, ..., \downarrow]^{\mathbf{S}})$. Let now $e_3 = \langle I_H, F\rangle$, where $F : T \to \downarrow$ is a folding (Proposition 4.10), then e_3 is an embedding from $H \times T$ into $H \times \downarrow$. Finally, $e = e_3 \circ e_2 \circ e_1$ gives us the desired embedding from *P* into the space-time algebra of *H*. □

6. CONCLUSIONS

Formalizing data dependencies as an algebraic structure gives rise to an algebra of data dependency operators and provides a framework for precise descriptions of the space-time embedding of software. Data dependency algebras, space-time algebras, and embeddings were in [7] seen as modular components for the parallel distribution of programs.

In this paper we have further studied the modularity properties of DDAs and embeddings. We have shown that they form a category which is complete (has all limits). This gives us many ways of building complex DDAs from simpler ones. It also allows us to factorize complicated space-time embeddings into simpler ones, and automatically build the complex embedding from the components.

This idea for parallel programming modularity can be extended to other properties of the categories of DDAs. We intend to follow this line of investigation by also studying the colimits of this category and the interaction between all the various kinds of dependency module components.

ACKNOWLEDGEMENTS

We thank Paul Taylor for providing the diagram package and the The Research Council of Norway for partial funding of this research.

REFERENCES

1. Perrin, G.-R. and Darte, A. (eds.). The data parallel programming model. *LNCS*, 1996, **1132**.
2. Lisper, B. Data parallelism and functional programming. In *The Data Parallel Programming Model* (Perrin, G.-R. and Darte, A., eds.). *LNCS*, 1996, **1132**, 220–251.
3. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. *MPI–The Complete Reference*. MIT Press, Cambridge, Mass., USA, 1996.
4. Squyres, J. M., Saphir, B. and Lumsdaine, A. The design and evolution of the MPI-2 C++ interface. In *Scientific Computing in Object-Oriented Parallel Environments* (Ishikawa, Y., Oldehoeft, R. R., Reynders, J. V. W. and Tholburn, M., eds.). *LNCS*, 1997, **1343**, 57–64.
5. Haveraaen, M. Data dependencies and space time algebras in parallel programming. Technical Report 45, Department of Informatics, University of Bergen, P.O.Box 7800, N-5020 Bergen, Norway, February 1990, revised 1997.
6. Čyras, V. and Haveraaen, M. Modular programming of recurrencies: a comparison of two approaches. *Informatica*, 1995, **6**, 397–444.
7. Haveraaen, M. An algebra of data dependencies and embeddings for parallel programming. *Form. Asp. Comput.* (forthcoming).
8. Haveraaen, M. Efficient parallelisation of recursive problems using constructive recursion. In *Euro-Par 2000 – Parallel Processing* (Bode, A., Ludwig, T., Karl, W. and Wismüller, R., eds.). *LNCS*, 2000, **1900**, 758–761.
9. Wolfe, M. (ed.). *High Performance Compilers for Parallel Computing*. Addison Wesley, Reading, Mass., 1996.

10. Thompson, B. C. and Tucker, J. V. Theoretical considerations in algorithm design. In *Fundamental Algorithms for Computer Graphics* (Earnshaw, R., ed.). Springer-Verlag, Berlin, 1985, 855–878.
11. Chen, M. C., Choo, Y. and Li, J. Crystal: Theory and pragmatics of generating efficient parallel code. In *Parallel Functional Languages and Compilers* (Szymanski, B. K., ed.). ACM Press, New York / Addison Wesley, Reading, Mass., 1991, 255–308.

Andmesõltuvusalgebrate ja -sisestuste kategooriast

Alexa Anderlik ja Magne Haveraaen

Paralleelarvuti programmeerimine kujutab endast sisuliselt andmesõltuvus-mustri sisestust arvuti aja–mälu-mustrisse. Üldiselt on see tehniliselt raske ülesanne. Nimetatud mustrite käsitlemine algebraliste struktuuridena – andmesõltuvusalgebrate ning aja–mälu-algebratena (viimased on esimeste erijuht) – annab konkreetsetest programmidest ja masinatest lahutatud vahendid formaalseks arutlemiseks sisestusprobleemi üle. Selles artiklis näitame, et andmesõltuvus-algebraid on võimalik tegurdada nõnda, et programmi sisestus paralleelarvutisse on jaotatav lihtsamate andmesõltuvusalgebrate sisestusteks lihtsamatesse aja–mälu-algebratesse. Need lihtsamad sisestused on seejärel automaatselt kombineeritavad programmi täielikuks sisestuseks masinasse.