Domains of Higher-Dimensional Automata

Eric Goubault

LIENS, Ecole Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, FRANCE,
email:goubault@dmi. ens.fr

Abstract. We carry on the program set up in [GJ92] by giving constructions of "domains of higher-dimensional automata (HDA)" on which we can define the truly-concurrent semantics of parallel languages, much in the style of domain theory (see [GS90]). In [GJ92] we gave a semantics for CCS-like languages. In this article, we show how to extend the technique to languages with real states, while keeping nice algebraic definitions. In particular, we are still able to compute local invariants which can decide a few computational properties of interest. For being used as actual computational definitions, the semantics is denotational (i.e. compositional); for being precise enough when it comes to studying the dynamic behaviour of programs, the denotations are higher-dimensional automata, which are no more than an operational behaviour (in the style of operational semantics, see [Plo81]). We conclude by giving the semantics of a small shared-memory imperative language.

1 Introduction

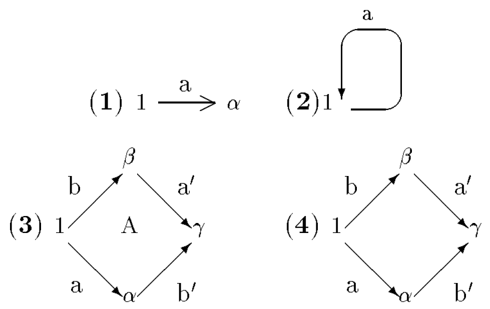
In [Pra91], Vaughan Pratt advocated a model of true concurrency based on geometric ideas. We presented in [GJ92] an elaboration on this idea, using basic homological algebra to formalize it. In this article, we recast this work in a slightly more general framework, enabling us to give a good account of the labelling and to get an analogous of events (see [Win88]) to fit in our setting. Then, we carry on by defining the computational properties which can be interpreted as properties of the geometry of HDA. Among them, we consider deadlocks, divergence and serialization. Techniques borrowed from homological algebra are used to extract the information, as in [GJ92]. We refer the reader to [Gou93] for seeing them at work with applications to a few branching-time semantic equivalences, generalizing results of [GJ92] about bisimulation equivalence. Then, we describe the categorical properties of higher-dimensional automata. It appears that they have a very rich structure: the category of HDA is complete, co-complete and is monoidal closed. This is enough for defining semantic domains in the style of domain theory, but this time, much more adapted to parallel languages. We end up by giving the semantics of a small shared-memory imperative language.

2 Higher-dimensional automata

2.1 Basic definitions

The unlabelled case Higher-dimensional automata are a natural generalization of "standard" automata (see [HU79]). They are built as sets of states and transitions between states, but also as sets of 2-transitions between transitions, and more generally n-transitions between (n-1)-transitions. Transitions (or 1-transitions) are usually depicted as segments, that is, one-dimensional objects, whereas states are just points, i.e. 0-dimensional objects. It is therefore natural to represent n-transitions as n-dimensional objects, whose boundaries are the (n-1)-transitions from which they can be fired, and to which they end up. n-transitions represent the concurrent execution of n sequential processes. Examples (1), (2) and (4) show HDA which are also standard automata. Example (4) is the interleaving of a and b, and expresses a different behaviour than example (3). Example (3) below shows a 2-transition A, which is the truly-concurrent execution of processes a and b.

Example 1.



But this 2-transition can be fired from a or from b at any time, thus the beginning of A is in some way a and b. Similarly, the end of A is a' and b'. One may want also to add coefficients (like integers) to transitions to keep track of the number of times we go through them. This motivates the introduction of free modules (or vector spaces when the coefficients form a field) generated by states and transitions, and source and target boundary operators acting on them.

Let then R (the coefficients) be a commutative integral and unitary ring (see [Lan84]). In general, for the sake of simplicity, R will be a field (in [GJ92], R was $\mathbb{Z}_2$) so that R-module is R-vector-space.

Definition 1. A (unlabelled) *higher dimensional automaton* (HDA) is a free R-module M with two gradings associated to two boundary operators, that is, consists of:

- a decomposition: M=$\Sigma_{p,q\in\mathbb{Z}} M_{p,q}$, such that $\forall p,q,\ M_{p,q}\cap(\Sigma_{r+s\neq p+q} M_{r,s})=0$.
- two differentials ∂_0 and ∂_1, compatible with the decomposition, giving M a structure of bicomplex:

$$\partial_0 : M_{p,q} \longrightarrow M_{p-1,q}$$
$$\partial_1 : M_{p,q} \longrightarrow M_{p,q-1}$$
$$\partial_0\circ\partial_0=0,\quad \partial_1\circ\partial_1=0,\quad \partial_0\circ\partial_1+\partial_1\circ\partial_0=0.$$

∂_0 is called the *source boundary operator* and ∂_1 is the *target boundary operator*. When we want to specify the domain and codomain of these boundary operators, we write $\partial_0^{p,q}$ for $\partial_0 : M_{p,q} \longrightarrow M_{p-1,q}$ and $\partial_1^{p,q}$ for $\partial_1 : M_{p,q} \longrightarrow M_{p,q-1}$. If $M_{p,q}\cap M_{p',q'}=\emptyset$ when (p,q)≠(p',q'), that is, when M is a free bigraded bidifferential R-module, then M is said to be an *acyclic HDA*. If M is a finite-dimensional module, then M is called a *finite state automaton*. For x in $M_{p,q}$, we say that x is of *dimension* p+q, denoted by *dim*x=p+q. Elements of dimension 0 are called *states*, elements of positive dimension n are *n-transitions*, elements of negative dimension n are *n-events*. First condition of definition 1 amounts to saying that elements of M have a unique dimension (states are distinct from n-transitions). Hence, for definitions which only involve a submodule of elements of a given dimension, we use an indexation by a unique integer, as in [GJ92].

Remark: A "standard" unlabelled automaton can be given the structure of a (unlabelled) higher-dimensional automaton. Let (A,Σ,δ,I,F) be an automaton; Q is a set of states, Σ is a set of transitions, δ is the transition relation, $\delta \subseteq \wp(A \times \Sigma \times A)$, I is the set of initial states, F is the set of final (or accepting) states. Define M by: M_0 is the free module generated by A, M_1 is the free module generated by Σ. Let D_F be the set $D_F = \{a \in A/\ \not\exists \sigma, a',\ (a,\sigma,a') \in \delta \ and\ a \notin F\}$ (it is the set of deadlocks of the automaton). Let $D_I = \{a' \in A/\ \not\exists \sigma, a\ (a,\sigma,a') \in \delta \ and\ a' \notin I\}$ ("false" initial states). Then,

$$\partial_0(\sigma) = a \Leftrightarrow_{def} (\exists a' \in A, \quad (a,\sigma,a') \in \delta) \wedge (a \in A \backslash D_I)$$
$$\partial_1(\sigma) = a' \Leftrightarrow_{def} (\exists a \in A, \quad (a,\sigma,a') \in \delta) \wedge (a' \in A \backslash D_F)$$

This construction projects all deadlocks onto 0 and all "false" initial states onto 0.

We picture the algebraic definition of the automata of example 1 (the symbol $\oplus$ denotes the direct sum of its two arguments, see [Lan84]):

(1)

$$\begin{tikzcd}
\mathrm{M}_{0,1} = (\mathrm{a}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,1} = (1) \arrow[d, "\partial_1"'] \\
\mathrm{M}_{0,0} = (\alpha) \arrow[r, "\partial_0"] & \mathrm{M}_{-1,0} = 0
\end{tikzcd}$$

with $\partial_0(a) = 1$ and $\partial_1(a) = \alpha$, is an acyclic finite state HDA. It comes from the standard automaton $(A, \Sigma, \delta, I, F)$ with A=$\{1, \alpha\}$, $\Sigma = \{a\}$, $\delta = \{(1, a, \alpha)\}$, $I = \{1\}$ and $F = \{\alpha\}$.

(2)

$$\begin{tikzcd}
\mathrm{M}_{0,1} = (\mathrm{a}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,1} = (1) \arrow[d, "\partial_1"'] \\
\mathrm{M}_{0,0} = (1) \arrow[r, "\partial_0"] & \mathrm{M}_{-1,0} = 0
\end{tikzcd}$$

with $\partial_0(a) = 1$ and $\partial_1(a) = 1$, is a finite state HDA which is not acyclic.

(3)

$$\begin{tikzcd}
\mathrm{M}_{1,1} = (\mathrm{A}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{0,1} = (\mathrm{a}) \oplus (\mathrm{b}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,1} = (1) \arrow[d, "\partial_1"'] \\
\mathrm{M}_{1,0} = (\mathrm{a}') \oplus (\mathrm{b}') \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{0,0} = (\alpha) \oplus (\beta) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,0} = 0 \arrow[d, "\partial_1"'] \\
\mathrm{M}_{1,-1} = (\gamma) \arrow[r, "\partial_0"] & \mathrm{M}_{0,-1} = 0 \arrow[r, "\partial_0"] & \mathrm{M}_{-1,-1} = 0
\end{tikzcd}$$

with $\partial_0(A) = a - b$, $\partial_1(A) = a' - b'$, $\partial_0(a) = \partial_0(b) = 1$, $\partial_1(a) = \partial_0(b') = \alpha$, $\partial_1(b) = \partial_0(a') = \beta$ and $\partial_1(a') = \partial_1(b') = \gamma$. It is an acyclic finite state HDA.

(4)

$$\begin{tikzcd}
 & \mathrm{M}_{0,1} = (\mathrm{a}) \oplus (\mathrm{b}) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,1} = (1) \arrow[d, "\partial_1"'] \\
\mathrm{M}_{1,0} = (\mathrm{a}') \oplus (\mathrm{b}') \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{0,0} = (\alpha) \oplus (\beta) \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & \mathrm{M}_{-1,0} = 0 \arrow[d, "\partial_1"'] \\
\mathrm{M}_{1,-1} = (\gamma) \arrow[r, "\partial_0"] & \mathrm{M}_{0,-1} = 0 \arrow[r, "\partial_0"] & \mathrm{M}_{-1,-1} = 0
\end{tikzcd}$$

with $\partial_0(a) = \partial_0(b) = 1$, $\partial_1(a) = \partial_0(b') = \alpha$, $\partial_1(b) = \partial_0(a') = \beta$ and $\partial_1(a') = \partial_1(b') = \gamma$. It is an acyclic finite state HDA.

Definition 2. A path (of length n) in a HDA M is a sequence of elements of B=$\{b_i\}$, a given basis of M ("elementary transitions"), p=$(p_i)_{0\leq i\leq n}$ such that:

$$p_0, p_n \in M_0 \; dim p_i \geq 0$$

$$\forall i, \partial_0(p_{i+1}) = \Sigma_j \alpha_j b_j, \quad with \quad \exists k, b_k = p_i \quad and \quad \alpha_k \neq 0 \quad or$$

$$\forall i, \partial_1(p_i) = \Sigma_j \alpha_j b_j, \quad with \quad \exists k, b_k = p_{i+1} \quad and \quad \alpha_k \neq 0$$

A n-dimensional path is a path whose elements are of dimension lower (or equal) than n. Paths in automaton (3) of example 1 are sub-paths of (1,a,α,b',γ), (1,b,β,a',γ), (1,a,A,b',γ), (1,a,A,a',γ), (1,b,A,a',γ) and (1,b,A,b',γ).

Definition 3. Let r,s be two integers. Let f be a function between two HDA $(M, \partial_0, \partial_1)$ and $(M', \partial'_0, \partial'_1)$, union of linear functions $f_{i,j}\colon M_{i,j} \rightarrow M'_{i+r,j+s}$ (f is bigraded). Then f is called a *morphism (of HDA) of degree (r,s)* if the $f_{i,j}$ verify:

$$\forall i, \forall x \in M_{i+1,j}, f_{i,j}(\partial_0(x)) = \partial'_0(f_{i+1,j}(x)).$$

$$\forall i, \forall x \in M_{i,j+1}, f_{i,j}(\partial_1(x)) = \partial'_1(f_{i,j+1}(x)).$$

A morphism of degree 0 will just be called a morphism (see [Lan84]).

Now, we have categories Υ of HDA with morphisms of degree 0, and a full subcategory Υ_a of acyclic HDA.

The labelling

Definition 4. A labelled HDA (over L) is a pair (M,l) composed of an unlabelled HDA M, and a morphism l: M$\longrightarrow$L. A morphism f: (M,l) $\longrightarrow$(M',l') of labelled HDA is a morphism of HDA between M and M' such that l'$\circ$f=l.

Hence the category of labelled HDA over L is the slice category (see [FS90]) Υ/L. The category of labelled acyclic HDA is the subcategory of Υ/L formed by the restriction of Υ to Υ_a. The action of l is to fold together all elements which have the same label. In most cases, all states are mapped onto a unique state of L.

Example 2. let L be the HDA such that $L_0 = (\mathbf{1})$, $L_1 = (\mathbf{a}) \oplus (\mathbf{b})$ with $\partial_j(\mathbf{a}) = \partial_j(\mathbf{b}) = \mathbf{1}$. Let M be the HDA of (4) of example 1. Define a module homomorphism l by l(a)=l(a')=**a**, l(b)=l(b')=**b** and l(1)=l(α)=l(β)=l(γ)=**1**. Then l is a morphism, and (M,l) is a labelled HDA over L.

2.2 Homology

Definition 5. For (Q,∂_0,∂_1) a HDA, we define two sequences of homology (see for instance [ML63]) modules [1]:

- $H_i(Q, \partial_0) = Ker\partial_0^i / Im\partial_0^i$
- $H_i(Q, \partial_1) = Ker\partial_1^i / Im\partial_1^i$

[1] Ker denotes the kernel of an application, and Im the image of an application.

An element of $Ker\partial_j^i$ is an *i-cycle*, and an element of $Im\partial_j^{i+1}$ is an *i-boundary*. An element of $H_i(Q,\partial_0)$ is called a *branching* of dimension i. An element of $H_i(Q,\partial_1)$ is called a *merging* of dimension i. We write $H_*(T,\partial_j)$ for $\bigoplus_{k\geq 0} H_k(T,\partial_j)$.

Example 3.
- For automaton (1) of example 1, we have: $H_0(M,\partial_0)=(\alpha)$, $H_0(M,\partial_1)=(1)$, and the other homology modules are null.
- For automaton (2), all the homology modules are null.
- For automaton (3), we have: $H_0(M,\partial_0)=(\gamma)$, $H_0(M,\partial_1)=(1)$, and the other homology modules are null.
- For automaton (4), we have: $H_0(M,\partial_0)=(\gamma)$, $H_0(M,\partial_1)=(1)$, $H_1(M,\partial_0)=(b-a)$, $H_1(M,\partial_1)=(b'-a')$, and the other homology modules are null. The branching (b-a) of dimension one expresses the fact that in (4) there is a non-deterministic choice between actions a and b. The merging (b'-a') shows that after the actions b' and a', the system goes to a same (idle) state.

If f: $(M,\partial_0,\partial_1)\rightarrow(M',\partial_0',\partial_1')$ is a morphism then f induces a module homomorphism $f_*: H_*(M,\partial_j)\rightarrow H_*(M',\partial_j')$ (j=0,1). This defines homology as a functor.

3 A few computational properties

We express here a few properties of interest in terms of homology, in the spirit of [GJ92]. We refer to [Gou93] for details.

3.1 Initial and final states, deadlocks, and divergence

We have already defined branchings and mergings in all dimensions. Branchings provide a measure of non-determinism in all dimensions, and distinguish the truly concurrent execution of processes from their (non-deterministic) interleaving. For instance, automaton (3) has no branching of dimension 1 whereas automaton (4) has one (the non-deterministic choice between a and b). Mergings "reduce" the non-determinism of an automaton. There are also interesting interpretations of elements of the homology modules in the lowest dimensions:

Definition 6. We call final (or accepting) state of an HDA M any elementary state generating $H_0(M,\partial_0)$.

Example 4. Consider the (standard) automaton (A,Σ,δ,I,F) with A=$\{u,v,w\}$, Σ=$\{a,b\}$, I=$\{u\}$, F=$\{v\}$ and δ=$\{(u,a,v),(u,b,w)\}$. Then, using our translation for standard automata, the associated HDA is M, with:

$$M_0=(u)\oplus(v)\oplus(w) \quad M_1=(a)\oplus(b)$$

and, $\partial_0(a)=u=\partial_0(b)$, $\partial_1(a)=v$, $\partial_1(b)=0$ Obviously, $H_0(M,\partial_0)=(v)=$ (thus F generates $H_0(M,\partial_0)$), $H_1(M,\partial_0)=(a)\oplus(b)$, $H_0(M,\partial_1)=(u)$ and $H_1(M,\partial_1)=(b)$.

Definition 7. We call initial state of an HDA M any state generating $H_0(M,\partial_1)$.

Consider the automaton of last example. We have seen that $H_0(M,\partial_1)=(u)$: u is the initial state of M. Now we come to deadlocks and "initial" deadlocks (or "false" initial states).

Definition 8. An elementary n-transition leading to (or is) a deadlock in a HDA M is a generator of $H_n(M, \partial_1)$. The word deadlock is given to the one transitions which deadlock M (for n greater than one, we say n-deadlock).

In example 4, b is a transition leading to a deadlock.

Definition 9. A n-transition leading to (or is) an initial deadlock in an HDA M is a generator of $H_n(M, \partial_0)$. Again, the word initial deadlock is given to 1-transitions.

If M does not have any initial deadlock then all states of M are reachable, and all transitions can be fired. In this case, we say that M is path-connected. Dually, if M does not have any deadlock, then all states are co-reachable. Then,

Definition 10. A HDA M diverges if and only if $H_0(M, \partial_0) = 0$ (it does not have any final state).

Automaton number 2 in example 1 diverges.

Dually, an HDA co-diverges if it has no beginning.

3.2 Serialization

A concurrent program is serializable if it "gives the same result" as a sequential execution of it. This is a highly geometric property for HDA: this means that all paths can be deformed continuously into another. More formally, we must define a notion of homotopy:

Definition and lemma 1 *An abstract 1-cycle of length m+n is an HDA described geometrically as:*

$$\begin{tikzcd}
 & \alpha_1 \arrow[r, "c_2"] & \ldots\alpha_{n-1} \arrow[dr, "c_n"] & \\
C_{n,m} = 1 \arrow[ur, "c_1"] \arrow[dr, "d_1"'] & & & \alpha_n \\
 & \beta_1 \arrow[r, "d_2"'] & \ldots\beta_{m-1} \arrow[ur, "d_m"'] &
\end{tikzcd}$$

Then,

Definition 11. An elementary 1-cycle of length m+n in a pointed HDA (Q,d) ($d \in Q_0$)[2] is a morphism $i : C_{n,m} \longrightarrow Q$ such that i(1)=d.

Let Q' be:

$$\begin{tikzcd}
 & t_3 & \\
t_1 \arrow[ur, "d"] \arrow[rr, phantom, "A"] & & t_2 \arrow[ul, "c'"'] \\
 & t_0 \arrow[ul, "c"] \arrow[ur, "d'"'] &
\end{tikzcd}$$

[2] d is generally chosen among the initial states

Let now f_1 and g_1 be the morphisms of HDA from $C_{2,2}$ to Q', defined by (prototypes of being "elementarily equivalent"):

$$f_1(c_1) = f_1(d_1) = c, \quad f_1(c_2) = f_1(d_2) = d$$

$$g_1(c_1) = g_1(d_1) = d', \quad g_1(c_2) = g_1(d_2) = c'$$

Definition 12. Let i: $C_{n,m} \longrightarrow P$ and j: $C_{n,m} \longrightarrow P$ be two elementary cycles in P. We say that i and j are elementarily equivalent (or that there exists an elementary transformation from i to j or from j to i) if and only if there exist (non null) morphisms r and s such that the following diagram is commutative:

$$\begin{tikzcd}
C_{n,m}/r(C_{2,2}) \arrow[r, "j"] & P/s(Q') & \\
C_{n,m} \arrow[u] \arrow[r, "i"] & P \arrow[u] & C_{n,m} \arrow[l, "j"'] \\
C_{2,2} \arrow[u, "r"'] \arrow[r, "f_1"] & Q' \arrow[u, "s"'] & C_{2,2} \arrow[u, "r"'] \arrow[l, "g_1"']
\end{tikzcd}$$

where the columns are exact.

r and s identify the difference between i and j (i=j on $C_{n,m}/r(C_{2,2})$). The difference is filled by a 2-transition (s(A)). This formalizes the idea that two cycles are equivalent if one can be deformed through 2-transitions into the other. The reflexive and transitive closure of the relation "is elementary equivalent to" is called homotopy. Homotopy then extends to paths with common beginnings and endings.

Definition 13. An unlabelled HDA M is serializable if and only if all one-dimensional paths of maximal length are homotopic.

This shows that M is serializable if and only if the "fundamental monoid" of M, i.e. the monoid of maximal paths modulo homotopy, is trivial. For instance, automaton number three in example 1 is serializable, whence automaton number four in example 1 is not. This extends naturally to the labelled case.

Definition 14. A labelled HDA (M,l) is serializable if and only if for all maximal one-dimensional paths p, q, there exists q' such that l(q)=l(q') and p is homotopic to q'

Obstructions to serializability in HDA can be found in $H_1(M, \partial_0)$, because this is precisely the presence of holes, as in automaton (4), which prevents the deformations of paths. We can also give a general notion of "serializability on n processors", obstructions of which can be found in $H_n(M, \partial_0)$. It generalizes the problem of serializability (n=1) to the following question: is a program "equivalent" to some implementation on a machine with n processors ?

4 Constructions

4.1 Limits and Colimits

0 is the zero object in categories Υ and Υ_a, that is, is both their initial and terminal object. We can also define cartesian products, coproducts, and equalizers and

coequalizers in Υ. Thus, Υ is finitely complete and finitely co-complete. In particular, amalgamated sums exist in Υ. The amalgamated sum of X and Y over Z is denoted by $X \coprod_Z Y$. Cartesian products correspond to some form of synchronized product. Coproducts are choice operators. Equalizers and coequalizers are used to model communication. To abbreviate the semantical constructions, we define a special operation on HDA generated by some module:

Definition and lemma 2 *Let A be a submodule of an HDA M. Then there exists a smallest sub-automaton of M containing A, denoted by Clos(A). We define an operation + on submodules of M, by:*

$$A + B = Clos(A) \coprod_{Clos(A) \cap Clos(B)} Clos(B).$$

+ corresponds to the geometrical operation of connected sum.

Direct limits and inverse limits (i.e. colimits - resp. limits - whose underlying diagram is a directed poset, see [FS90]) exist in the category of modules (see [Lan84]). This entails that they exist in Υ and Υ_a. Therefore,

Proposition 15. *Υ is complete and co-complete.*

4.2 Tensor product and Hom functor

Definition 16. Let M_1 and M_2 be two HDA. Define a R-module T by: $T_{p,q} = \Sigma_{k,l} M_{1,p-k,q-l} \otimes M_{2,k,l}$ and two operators (j=0,1) by $\partial_j(x \otimes y) = \partial_j(x) \otimes y + (-1)^{(dim\, x)} x \otimes \partial_j(y)$, that is,

$$\partial_j^{n,m} = \Sigma_{p+r=n, q+s=m} (\partial_j^{p,q}[M_1] \otimes Id + (-1)^{p+q} Id \otimes \partial_j^{r,s}[M_2])$$

Then T is a HDA.

This construction is the tensor product of the two complexes associated with M_1 and M_2 (see [Mas78]). The reader can verify that it is actually a tensor product in the category Υ (see for instance [FS90]. It corresponds to the truly parallel composition of processes with no communication.

We define the action of F on morphisms f: X $\longrightarrow$ Y by:

$$F(f) : F(X) \longrightarrow F(Y), \quad F(f)(x \otimes m) = f(x) \otimes m$$

F(f) is a morphism, because $\partial_i(f(x) \otimes m) = \partial_i(f(x)) \otimes m + (-1)^{dim\, f(x)} f(x) \otimes \partial_i(m) = F(f)(\partial_i(x) \otimes m + (-1)^{dim\, x} x \otimes \partial_i(m)) = F(f)(\partial_i(x \otimes m))$. Therefore, for M an HDA, $(. \otimes M)$ and $(M \otimes .)$ are endofunctors on Υ and Υ_a.

Definition and lemma 3 *Let P and Q be two R-bicomplexes. We define Hom(P,Q) as the HDA whose objects of bidegree (r,s) are:*

$$Hom(P,Q)_{r,s} = \prod_{i,j \in \mathbb{Z}} Hom(P_{i,j}, Q_{i+r,j+s})$$

(where $Hom(P_{i,j}, Q_{i+r,j+s})$ is the R-module of R-linear maps from $P_{i,j}$ to $Q_{i+r,j+s}$) and whose boundary operators are:

$$\partial_i(f_{j,k}) = \partial_i[Q] \circ f_{j,k} - (-1)^{r+s} f_{j,k} \circ \partial_i[P]$$

for i=0,1, and $f_{j,k}$ being the (j,k) component of some f in $Hom(P,Q)_{r,s}$.

Now, Hom is a contravariant functor in its first argument, and a covariant one in its second argument. Let f: P $\longrightarrow$ P' be a morphism of HDA. Then Hom(f,Q): Hom(P',Q) $\longrightarrow$ Hom(P,Q) is the morphism such that Hom(f,Q)(h)=h$\circ$f. If g: Q $\longrightarrow$ Q' is a morphism, then Hom(P,g): Hom(P,Q) $\longrightarrow$ Hom(P,Q') is the morphism defined by Hom(P,g)(h)=g$\circ$h.

Proposition 17 (Monoidal closedness).

$$Hom(P \otimes Q, R) \cong Hom(P, Hom(Q, R)) \quad \Upsilon(P \otimes Q, R) \cong \Upsilon(P, Hom(Q, R))$$

and the map eval: Hom(P,Q)$\otimes$P $\rightarrow$ Q with eval($\{f_i\}\otimes x$)=$f_{dim x}(x)$ is a morphism.

For f an object of dimension n of Hom(P,Q), we say that:
if n>0 then f *increases the degree of parallelism* (by n) and
if n<0 then f *decreases the degree of parallelism* (by n).

Example 5. Let a be an elementary object of dimension n in a HDA M. Suppose $f_a(x) = a \otimes x$ is well defined on M. Then dimf=n and $\partial_j(f) = f_{\partial_j(a)}$.

Let (1) be the HDA generated by a state 1, and with null boundary operators.

Definition 18. Let M be an HDA. Then the HDA $M^* = Hom(M, (1))$ is called the dual of M. Elements of M^* are called functionals.

Let $< ., . >$ be the bilinear form defined on M by $< x, y >= y^*(x)$. We come now to the description of the dual of an HDA.

Lemma 19. *Let M be a finite state automaton with basis B=$\{b_i\}$. Then M^* has basis B^*, $dimb^* = -dimb$. Moreover, if the boundary operators of M^* are denoted by ∂_0^* and ∂_1^*, then:*

$$\forall x, y \in B, < \partial_i(x), y >=< x^*, \partial_i^*(y^*) >$$

where $< x, y >$ denotes the usual inner product $< x, y >= \Sigma_i x_i y_i$, if $x = \Sigma_i x_i b_i$ and $y = \Sigma_i y_i b_i$.

The operators ∂_0^* and ∂_1^* are respectively called the end boundary operator and the start boundary operator. Notice that for M a finite state automaton, M and M^{**} are isomorphic (i.e. this is a duality). By analogy with [Pra92], the dual of transitions, which bear information and change time, are events, which bear time and change information. Thus, the (standard) boundary operators describe the temporal beginning and ending of events, whereas the dual boundary operators describe the information we have at the source, and the information we have at the target.

Example 6. The dual of automaton (1) in example 1 is:

$$\begin{tikzcd} M_{0,1} = 0 \arrow[r, "\partial_0"] \arrow[d, "\partial_1"'] & M_{-1,1} = (1^*) \arrow[d, "\partial_1"'] \\ M_{0,0} = (\alpha^*) \arrow[r, "\partial_0"] & M_{-1,0} = (a^*) \end{tikzcd}$$

In this HDA, we have one 1-event namely a^*, "waiting for transition a to be fired". The "information" beginning of 1^* is a^*, the "information" end of α^* is a^*.

Homology behaves quite well with respect to colimits and tensor products. In particular, it commutes with direct sums and direct limits, and commutes in most cases (for instance when R is a vector space) with the tensor product. For more general amalgamated sums, we have to use the Mayer-Vietoris sequence (see [ML63]). Finally, for the Hom functor, we have the universal coefficient theorem. All these techniques, we have no time to discuss here enable us to compute the different invariants of section 3 in an inductive manner, for each application of the different constructions seen above.

5 Semantic domains

In this section, we wish to give a compositional (or denotational) semantics for some parallel languages, with denotations being HDA, that is, describing the "higher-dimensional" traces of programs.

5.1 General principles – ground domains

First, we have to define domains for values, and define real states.

Let V be a set (of values). We write $\overline{V}$ for the HDA whose states are generated by V, and whose boundary operators are null. We have already seen an example of this construction, for V=$\{1\}$; $\overline{V}$ was written (1). This construction will be applied for sets of values like $\mathbb{N}$, or Bool=$\{tt, ff\}$. Notice that it is again a functorial construction: if f is a function between two sets V and V', then $\overline{f}$ (sometimes abbreviated by f), the linear extension of f, is a morphism of degree 0 between $\overline{V}$ and $\overline{V'}$.

The same construction can be carried out for any relation on sets of values. For instance, let us consider a relation R(x,y) on V×V. Construct a "relation" $\overline{R}$ on $\overline{V}\otimes\overline{V}$, with value in $\overline{Bool}$ by $\overline{R}(\overline{x},\overline{y}) =_{def} \overline{R}(x\otimes y) = (tt) \Leftrightarrow R(x,y)$ and $\overline{R}(\overline{x},\overline{y}) =_{def} \overline{R}(x\otimes y) = (ff) \Leftrightarrow \neg R(x,y)$.

Now, suppose we have elementary functions on these sets of values. We want to represent the application of such a function f to a value by a 1-transition between the input value to the output value. The way to do this, is to construct (when it exists in the considered domains) the $(\partial_1 - \partial_0)$-chain homotopy (see [ML63]) linking the input to the output state, that is the transition between Id and $\overline{f}$, denoted by [Id,f] or $\check{f}$ ("name of f"):

Suppose we have a domain D of HDA (elements of which are its sub-automata) containing 1-transitions s and t with $\partial_0(s) = 1$, $\partial_1(s) = 0$, and $\partial_0(t) = 0$, $\partial_1(t) = 1$. Let f and g be two linear maps, and define:

$$[f, g] = s \otimes f + t \otimes g$$

Then,

$$\partial_0 \circ [f, g] = f - [\partial_0 \circ f, \partial_0 \circ g] \quad \partial_1 \circ [f, g] = g - [\partial_1 \circ f, \partial_1 \circ g]$$

and when f and g are morphisms (of any degree),

$$\partial_0 \circ [f, g] + [f, g] \circ \partial_0 = f \quad \partial_1 \circ [f, g] + [f, g] \circ \partial_1 = g$$

Thus,

$$(\partial_1 - \partial_0) \circ [f, g] + [f, g] \circ (\partial_1 - \partial_0) = g - f$$

and [f,g] is an $(\partial_1 - \partial_0)$-chain homotopy between f and g. To come back to our function f, $\check{f} = [Id, f]$, then: $\partial_0(\check{f}) = Id$ and $\partial_1(\check{f}) = \overline{f}$. We have also, $\forall x \in \overline{V}$,

$\partial_0(\check{f}(x)) = x$ and $\partial_1(\check{f}(x)) = \overline{f}(x)$. Hence we call $\check{f}$ "name of f", because it is the label of all applications of f.

In general, we have to use fresh copies of these t and v to build homotopies. These copies will be denoted by t_i and v_i, where i is an index (generally in $\mathbb{N}$). For f a linear function on a HDA V, we define an extension of f on tensor products of t_i, v_i and elements of V by:

$$f(t_i) = t_i \quad f(v_i) = v_i$$
$$f(x \otimes y) = f(x) \otimes f(y)$$

If f is a morphism (of degree 0) on V, then this extension defines a morphism as well. With these conventions, we have the following laws of calculus:

$$f[g,h] = [fg, fh] \quad [g,h]f = [gf, hf]$$
$$f\check{g} = [f, fg] \quad \check{g}f = [f, gf]$$
$$[f,g][k,l] = [[fk, fl], [gk, gl]]$$

The last equation shows that homotopies compose to give homotopies of higher dimension. We define also for a linear function f (not necessarily a morphism), another linear function, $\hat{f}$, called the sequentialization of f, by:

$$\hat{f}(g) = g + f \circ H_0(g, \partial_0)$$

We will see its use further on.

5.2 Recursive domain equations

Proposition 20. *Let M be an HDA. Functors $(. \otimes M)$, $(M \otimes .)$, $Hom(., M)$ and $(M+.)$ are ω-continuous, that is, preserve direct limits. $Hom(M, .)$ transforms direct limits into inverse limits.*

By standard results (see for example [AL91]), we know, using proposition 15, that if we have G such that $\forall M \in \Upsilon$, G(M,.) and G(.,M) are ω-continuous functors. Then,

$$\exists D \in \Upsilon, \quad D \equiv M + G(D, D)$$

Generally, we construct HDA as sub-HDA of a huge one, described by a recursive equation. This gives us also a means to label HDA, just knowing the labels of the "atomic actions". Let (M,l) be a labelled HDA over L. Consider now the equation D=M+G(D,D), where G is ω-continuous in each argument. There exists a solution D to this equation. Let now D_L be the HDA verifying the equation D_L=L+G(D_L,D_L). The solution to this equation is given by a limit of a diagram (see [AL91]). l induces a morphism of the diagram defining D and the one defining D_L. Its limit, still called l is the induced labelling of D over L.

Example 7. Let D be an HDA verifying D=M+D$\otimes$D (D is the envelopping algebra of M) where M is the HDA:

$$M_0 = (1) \oplus (\alpha) \oplus (\alpha') \oplus (\beta) \oplus (\beta') \; M_1 = (a) \oplus (a') \oplus (b) \oplus (b')$$

with $\partial_0(a) = \partial_0(a') = \partial_0(b) = \partial_0(b') = 1$ and $\partial_1(a) = \alpha$, $\partial_1(a') = \alpha'$, $\partial_1(b) = \beta$, $\partial_1(b') = \beta'$. Let L be the one defined in example 2. M can be labelled over L by l(a)=l(a')=**a**, l(b)=l(b')= **b**, l(1)=l(α)=l(α')=l(β)=l(β')=**1**. Then it extends to a labelling of D over D_L. For instance, l(a$\otimes\beta$)=**a**, or l($\alpha'\otimes$a$\otimes$b)= **a**$\otimes$**b**.

6 Example - A toy imperative language

Let $\mathcal{L}$ be the language (first-order imperative language - shared memory) whose syntax is defined as follows:
Let Var be a set of variable names (x, y, z...). We consider a set of values v $\in$ Val, containing integers n, booleans *tt* and *ff*. We write X, Y, Z for objects which are values or variables. f is any function on Val.
The language is formed out of values v, tests t, and expressions e:

$$\begin{array}{ll} v ::= x & e ::= nil \\ \quad \mid\ n & \quad \mid\ x := v \\ \quad \mid\ tt & \quad \mid\ x := h(x,v) \\ \quad \mid\ ff & \quad \mid\ e; e' \\ t ::= R(X,Y) & \quad \mid\ e \mid e' \\ & \quad \mid\ (t \rightarrow e)\Box(t' \rightarrow e') \\ & \quad \mid\ recx.q(x) \end{array}$$

where x:=h(x,v) is a function like (x:=.), (x$\times$=.), or (x+=.) etc. That is, proceeds to an "atomic" operation on a variable. q is any syntactic expression of $\mathcal{L}$ with one hole (a context). Now, we give the semantics, considering the following domains:
C is the HDA with C_0 generated by Var and $\partial_0 = \partial_1 = 0$. V is the HDA with V_0 generated by Val and $\partial_0 = \partial_1 = 0$. But, now, we would like to have environments (i.e. assignments of values to variables) as states of our automata: the domain D to be defined should include Env=Hom(C$\oplus$V,C$\oplus$V)[3]. But we want also to have all homotopies (all transitions of any dimension) between states. This requires for D to have all tensor products between the t_i, v_i and elements of D. This leads to defining D by the equation:

$$D \equiv (t_{c,i})_{c,i} \oplus (v_{c,i})_{c,i} \oplus Hom(C \oplus V, C \oplus V) \oplus C \oplus V + (D \otimes D)$$

where c is an index lying in the set $\{x := n, x := h(x,v)/x \in Var,\ v \in Val\}$. The domain for the labelling is defined by:

$$D_L \equiv (t_c)_c \oplus (v_c)_c \oplus Hom(C \oplus V, C \oplus V) \oplus C \oplus V + (D_L \otimes D_L)$$

The labelling l is induced by $l(t_{c,i}) = t_c, \quad l(v_{c,i}) = v_c$. Define now the function $[u \Leftarrow v]$ (an elementary substitution) on C$\oplus$V by:

$$[u \Leftarrow v](u) = v, \quad [u \Leftarrow v](w) = w$$

for all w$\neq$u. Therefore, $[u \Leftarrow v] = v \otimes u^* + \Sigma_{w \neq u} w \otimes w^*$ It can be extended to a morphism on D as described in the previous section.
The functions h considered in L induce morphisms of the form h_x from ρ $\in$Env to Env:

$$h_x(\rho,v)(x) = h(\rho(x),v) \quad h_x(\rho,v)(y) = \rho(y)$$

for all y$\neq$x. Their action is to apply the arithmetic function which h describes to the only x part of the substitution ρ. For instance, (x:=.) induces the morphism [x $\Leftarrow$

[3] Env may also be called domain of substitutions, or store. Valid substitutions are always identity on values.

.]. In the case of x+=v, that is h(x,v)=x+v, we have for example, $h_x([x \Leftarrow u][y \Leftarrow w], v) = [x \Leftarrow u+v][y \Leftarrow x]$.

Then we have a semantic function $[\![.]\!]: \mathcal{L} \longrightarrow Hom(Env, D)$ given by:
for values,

$$[\![x]\!]\rho = \rho(x) \tag{1}$$

$$[\![n]\!]\rho = n \tag{2}$$

$$[\![tt]\!]\rho = tt \tag{3}$$

$$[\![ff]\!]\rho = ff \tag{4}$$

for tests,

$$[\![R(X,Y)]\!]\rho = \overline{R}([\![X]\!]\rho, [\![Y]\!]\rho) \tag{5}$$

for processes,

$$[\![nil]\!]\rho = \rho \tag{6}$$

$$[\![x := v]\!]\rho = [\rho, \rho \circ [x \Leftarrow [\![v]\!]\rho]] = \rho \circ [x \Leftarrow \check{[\![v]\!]}\rho] \tag{7}$$

$$[\![x := h(x,v)]\!]\rho = [\rho, h_x(\rho, [\![v]\!]\rho)] = h_x(., \check{[\![v]\!]}\rho)\rho \tag{8}$$

$$[\![e; e']\!]\rho = \hat{[\![e']\!]}([\![e]\!]\rho) \tag{9}$$

$$[\![e \mid e']\!]\rho = [\![e]\!]([\![e']\!]\rho) + [\![e']\!]([\![e]\!]\rho) \tag{10}$$

$$[\![(t \to e)\Box(t' \to e')]\!]\rho = tt^*([\![t]\!]\rho).[\![e]\!]\rho + tt^*([\![t']\!]\rho).[\![e']\!]\rho \tag{11}$$

$$[\![recx.q(x)]\!]\rho = \varinjlim [\![q^n(nil)]\!]\rho \tag{12}$$

In all these equations, the labelling is implicit: when an homotopy is used for the semantics of x:=v or x:=h(x,v), it is formed of some fresh $t_{x:=v,i}$ and $v_{x:=v,i}$ or $t_{x:=h(x,v),i}$ and $v_{x:=h(x,v),i}$ respectively.

Equations 1, 2, 3, 4 and 5 are obvious. Equation 6 reads "nil does not act on the environment". Equations 7 and 8, written in two forms, build an homotopy between the environment and the transformed one (notice that substitutions compose the other way round). When ρ is a state, this is just a transition from the input of h to the output of h. Equation 9 applies the sequentialization of $[\![e']\!]$ to $[\![e]\!]$, that is, applies e' to the final states of e, and takes the union with the translation of e.

Equation 10 looks like interleaving, but is not. $[\![e']\!]([\![e]\!]\rho)$ is isomorphic to $([\![e']\!]) \otimes ([\![e]\!]\rho)$, thus is a good candidate as a parallel composition (see [GJ92]). But $[\![e]\!]$ and $[\![e']\!]$ may not commute if some of their actions are not independant, therefore we need the term $[\![e]\!]([\![e']\!]\rho)$. There can be non-independance if there is simultaneous use of the shared memory. Equation 11 takes the (disjoint or not) union of the two alternatives of the guarded statement. Finally, equation 12 takes the unfolding of a recursive agent as its semantics. The unfolding is represented by the direct limit of the diagram whose objects are the successive steps of unfolding $[\![q^n(nil)]\!]\rho$, and whose morphisms are the obvious ones (all morphisms between $[\![q^{n-1}(nil)]\!]\rho$ and $[\![q^n(nil)]\!]\rho$).

Example 8. – We consider the term x+=1 in the context $\rho = [x \Leftarrow 1]$:

$$[\![x+=1]\!]\rho = [\rho, h_x(\rho, 1)]$$

$$= [[x \Leftarrow 1], [x \Leftarrow 2]]$$

$$[\![x+=1]\!]\rho = [x \Leftarrow 1] \xrightarrow{\text{x}+=1} [x \Leftarrow 2]$$

– Now, consider the term (x:=1)|(x+=1) in the context $\rho = [x \Leftarrow 0] \circ [y \Leftarrow 42]$:

$$[\![(x := 1) \mid (x+ = 1)]\!]\rho = [\![(x := 1)]\!]([\![(x+ = 1)]\!]\rho) + [\![(x+ = 1)]\!]([\![(x := 1)]\!]\rho)$$

But,

$$[\![(x := 1)]\!]\rho = \rho \circ [x \check{\Leftarrow} 1]$$

$$= [\rho, [x \Leftarrow 1] \circ [y \Leftarrow 42]]$$

Then,

$$[\![(x+ = 1)]\!]([\![(x := 1)]\!]\rho) = [([\![(x := 1)]\!]\rho), h_x(([\![(x := 1)]\!]\rho, 1)]$$

$$= [[[x \Leftarrow 0][y \Leftarrow 42], [x \Leftarrow 1][y \Leftarrow 42]], [[x \Leftarrow 1][y \Leftarrow 42], [x \Leftarrow 2][y \Leftarrow 42]]]$$

which is geometrically realized by a square whose four vertices (only three of them are disjoint) are $[x \Leftarrow 0][y \Leftarrow 42]$, $[x \Leftarrow 1][y \Leftarrow 42]]$, $[x \Leftarrow 1][y \Leftarrow 42]]$, and $[x \Leftarrow 2][y \Leftarrow 42]$.

– Let us compute the semantics of (x:=1);(x+=3) in the context $\rho = [x \Leftarrow 0]$:

$$[\![(x := 1); (x+ = 3)]\!]\rho = [\![(x+ \hat{=} 3)]\!]([\![x := 1]\!]\rho)$$

$$= [\![(x+ \hat{=} 3)]\!]([[x \Leftarrow 0], [x \Leftarrow 1]])$$

$$= [\![(x+ = 3)]\!]([x \Leftarrow 1]) + [[x \Leftarrow 0], [x \Leftarrow 1]]$$

$$= [[x \Leftarrow 1], [x \Leftarrow 4]] + [[x \Leftarrow 0], [x \Leftarrow 1]]$$

$$[\![(x := 1); (x+ = 3)]\!]\rho \; = {\scriptstyle [x \Leftarrow 0]} \xrightarrow{\text{x = 1}} {\scriptstyle [x \Leftarrow 1]} \xrightarrow{\text{x+ = 3}} {\scriptstyle [x \Leftarrow 4]}$$

– Finally, let e=((x=1?→y:=2)□(x=y?→x:=0)). Then in context $\rho = [x \Leftarrow 1][y \Leftarrow 1]$ we have:

$$[\![e]\!] = tt^*([\![x = 1?]\!]\rho).[\![y := 2]\!]\rho + tt^*([\![x = y?]\!]\rho).[\![x := 0]\!]\rho$$

But,

$$[\![x = 1?]\!]\rho = tt$$

$$[\![x = y?]\!]\rho = tt$$

thus,

$$[\![e]\!] = [\![y := 2]\!]\rho + [\![x := 0]\!]\rho$$

$$= [[x \Leftarrow 1][y \Leftarrow 1], [x \Leftarrow 1][y \Leftarrow 2]] + [[x \Leftarrow 1][y \Leftarrow 1], [x \Leftarrow 0][y \Leftarrow 1]]$$

Thus,

$$\begin{tikzcd}[\![e]\!]\rho \; = {\scriptstyle [x \Leftarrow 1][y \Leftarrow 1]} \arrow[ur, "\text{y := 2}"] \arrow[dr, "\text{x := 0}"'] & {\scriptstyle [x \Leftarrow 1][y \Leftarrow 2]} \\ & {\scriptstyle [x \Leftarrow 0][y \Leftarrow 1]}\end{tikzcd}$$

This is a one-dimensional branching at state $[x \Leftarrow 1][y \Leftarrow 1]$. It describes an internal non-deterministic choice.

7 Conclusion and Future Work

We have presented in this article, the basis for a semantic theory of true concurrency which gather techniques from operational and denotational semantics. We believe that the rich algebraic structure of HDA and the many techniques available for computing their properties make them interesting denotations for programs.

The technique, briefly exemplified here, can certainly be used for parallel functional languages. The domain to be used should include all contexts, not limited to the first-order case as in our example. But the properties of the functor Hom are all we need for that (see [Gou93] for a parallel lambda calculus). All standard techniques from denotational semantics can also be fruitfully applied. For instance, continuations give meanings to fork operators (see [Gou93] for an example), and not just to ordinary parallel operators.

Finally, we think that this model shows good promise for giving semantics to real-time parallel languages. A well known adjunction between simplicial complexes and topological spaces of the homotopy type of CW-complexes can most probably inspire this attempt.

Acknowledgements We thank Vaughan Pratt, Samson Abramsky and Patrick Cousot for their support, since the very beginning of our work, Thomas Jensen, Regis Cridlig and Bruno Monsuez for their very valuable help. Finally, we would like to thank Jeremy Gunawardena with whom we had a very nice discussion, in particular about serialization, at Concur'92.

References

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, types and structures.* The MIT Press, second edition, 1991.

[FS90] Peter J. Freyd and Andre Scedrov. Categories, allegories. In *North-Holland Mathematical Library*, volume 39. North-Holland, 1990.

[GJ92] Eric Goubault and Thomas P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[Gou93] Eric Goubault. Higher-dimensional automata. Technical report, Ecole Normale Supérieure, to appear 93.

[GS90] C.A. Gunther and D.S. Scott. Semantic domains. In *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison–Wesley, 1979.

[Lan84] Serge Lang. *Algebra.* Addison Wesley, second edition, 1984.

[Mas78] William S. Massey. Homology and cohomology theory. In *Monographs and Textbooks in Pure and Applied Mathematics*, number 46. Marcel DEKKER, INC., 1978.

[ML63] Saunders Mac Lane. Homology. In *Die Grundlehren der Mathematishen Wissenschaften in Einzeldarstellungen*, volume Band 114. Springer Verlag, 1963.

[Plo81] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus, 1981.

[Pra91] Vaughan Pratt. Modeling concurrency with geometry. In *Proc. 18th ACM Symposium on Principles of Programming Languages*. ACM Press, 1991.

[Pra92] Vaughan Pratt. The duality of time and information. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[Win88] Glynn Winskel. An introduction to event structures. *Lecture notes in computer science*, (354), 1988.