Math. Struct. in Comp. Science (2000), *vol.* 10, *pp.* 411–425. Printed in the United Kingdom
© 2000 Cambridge University Press

Geometry and concurrency: a user's guide

ERIC GOUBAULT[†]

LETI (CEA - Technologies Avancées)
DEIN-SLA-CEA F 91191 Gif-sur-Yvette Cedex, France
Email: Eric.Goubault@cea.fr

Received 1 December 1999; revised 2 February 2000

Geometrical methods in concurrency theory (and in distributed systems theory) have appeared recently for modelling and analyzing the behaviour of systems and also for solving computability and complexity issues. We identify some of the main directions of research, and survey some of the major ideas on which all this is based (some of which are now more than thirty years old).

1. Introduction

'Geometry and Concurrency' is an area of research consisting of techniques that use geometrical reasoning to describe and solve problems appearing in concurrent systems. Here geometrical reasoning means mostly using techniques from algebraic topology[‡], and therefore involving an idea of 'topological invariance' under some form of homotopy, as will be explained in the next section. Graphs and partial orders do indeed include some form of geometric reasoning, but these are now standard techniques in computer science, to which quite a few conferences and journals are devoted, whereas we will be more interested in 'higher-dimensional' phenomena here.

These techniques have cropped up in different areas of computer science for over thirty years now. We have somewhat artificially subdivided them into three main strands. One is concerned with giving semantics to concurrent machines and languages, for formalizing and analyzing their properties. Another has been mostly concerned with distributed databases and the scheduling of transactions (with a view to their correctness). Finally, the most recent strand is in the field of fault-tolerant protocols for distributed systems, where the main application is the determination of the computability (and then complexity) of some 'functions' on given distributed architectures. After a brief history in Section 2, we review the main contributions to date in all three directions. In Sections 3, 4 and 5, we introduce the articles presented in this special issue of Mathematical Structures in Computer Science, and describe how they fall naturally into one of these pre-existing

[†] Part of this work was done when the author was at Ecole Normale Supérieure, Paris.
[‡] More or less abstract, from simple topological notions like connectedness (Spanier 1966), simplicial complexes (May 1967; Gabriel and Zisman 1967), *etc.* to abstract homotopy categories (Baues 1989).

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

strands. Finally, in Section 6 we give a few directions for future work, and in particular for some areas of research for which these techniques seem also to have appeared or might be useful.

2. A brief history

The first 'algebraic topological' model that I am aware of is called a *progress graph* and has appeared in operating systems theory, in particular, for describing the problem of 'deadly embrace'[†] in 'multiprogramming systems'. *Progress graphs* were introduced in Coffman *et al.* (1971), but attributed there to E. W. Dijkstra. In fact they also appeared slightly earlier (for editorial reasons it seems) in Shoshani and Coffman (1970) The basic idea is to give a description of what can happen when several processes are modifying shared ressources. Given a shared resource a, we see it as its associated semaphore, which rules its behaviour with respect to processes. For instance, if a is an ordinary shared variable, it is customary to use its semaphore to ensure that only one process at a time can write on it (this is mutual exclusion). Then, given n deterministic sequential processes $Q_1, \ldots, Q_n$, abstracted as a sequence of locks and unlocks on shared objects, $Q_i = R^1 a_i^1 . R^2 a_i^2 \cdots R^{n_i} a_i^{n_i}$ (R^k being P or V[‡]), there is a natural way to understand the possible behaviours of their concurrent execution, by associating to each process a coordinate line in $\mathbb{R}^n$. The state of the system corresponds to a point in $\mathbb{R}^n$, whose ith coordinate describes the state (or 'local time') of the ith processor.

Consider a system with finitely many processes all running together. We assume that each process starts at (local time) 0 and finishes at (local time) 1; the P and V actions correspond to sequences of real numbers between 0 and 1, which reflect the order of the P's and V's. The initial state is $(0, \ldots, 0)$ and the final state is $(1, \ldots, 1)$. An example consisting of the two processes $T_1 = Pa.Pb.Vb.Va$ and $T_2 = Pb.Pa.Va.Vb$ gives rise to the two dimensional *progress graph* of Figure 1.

The shaded area represents states that are not allowed in any execution path, since they correspond to mutual exclusion. Such states constitute the *forbidden area*. An *execution path* is a path from the initial state $(0, \ldots, 0)$ to the final state $(1, \ldots, 1)$ avoiding the forbidden area and increasing in each coordinate – time cannot run backwards. We call these paths *directed paths* or dipaths. This entails that paths reaching the states in the dashed square underneath the forbidden region, marked 'unsafe' are deemed to deadlock, that is, they cannot possibly reach the allowed terminal state, which is $(1, 1)$ here. Similarly, by reversing the direction of time, the states in the square above the forbidden region, marked 'unreachable', cannot be reached from the initial state, which is $(0, 0)$ here. Also notice that all terminating paths above the forbidden region are 'equivalent' in some sense, given that they are all characterized by the fact that T_2 gets a and b before T_1 (as far as resources are concerned, we call this a *schedule*). Similarly, all paths below the forbidden region are characterized by the fact that T_1 gets a and b before T_2 does.

[†] This was the term originally used by E. W. Dijkstra (Dijkstra 1968), but it is now more usually called deadlock.
[‡] Here we use E. W. Dijkstra's notation P and V (Dijkstra 1968) for, respectively, acquiring and releasing a lock on a semaphore.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

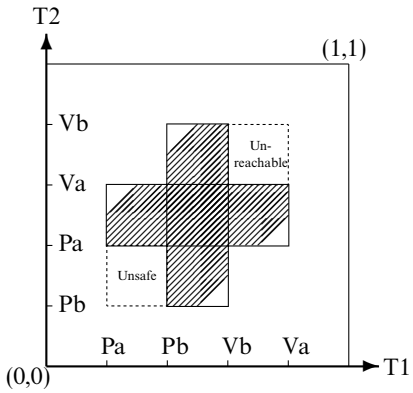


Fig. 1. Example of a progress graph

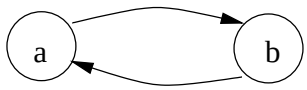


Fig. 2. The corresponding request graph

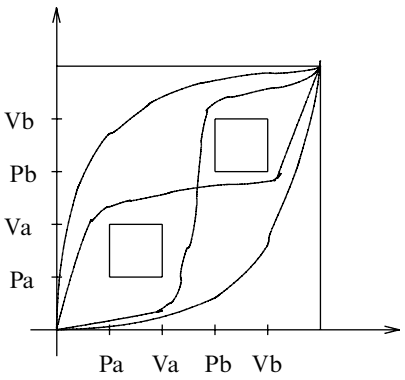


Fig. 3. The progress graph corresponding to $Pa.Va.Pb.Vb \mid Pa.Va.Pb.Vb$

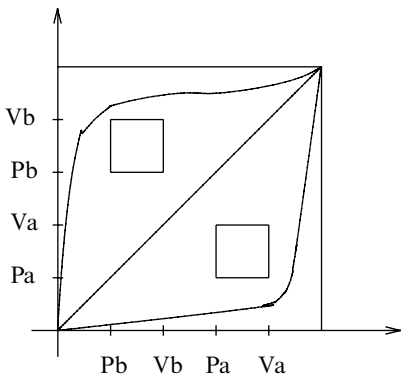


Fig. 4. The progress graph corresponding to $Pb.Vb.Pa.Va \mid Pa.Va.Pb.Vb$

On this picture, one can already recognize many ingredients that are at the centre of the main problem of algebraic topology, namely the classification of shapes modulo 'elastic deformation'. As a matter of fact, the actual coordinates that are chosen for representing the times at which Ps and Vs occur are unimportant, and these can be 'stretched' in any manner, so the properties (deadlocks, schedules, *etc.*) are invariant under some notion of deformation, or *homotopy*. This is a particular kind of homotopy though, and this will be at the centre of many difficulties in later work. We call it (in subsequent work) *directed homotopy* or *dihomotopy* in the sense that it should preserve the direction of time. For instance, the two homotopic shapes of Figure 3 and Figure 2, both of which have two holes, have a different number of dihomotopy classes of dipaths. In Figure 3 there are essentially four dipaths up to dihomotopy (that is, four schedules corresponding to all possibilities of accesses of resources a and b), whereas in Figure 2 there are essentially three dipaths up to dihomotopy.

There is another method for determining deadlocks in such situations, which was, of

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

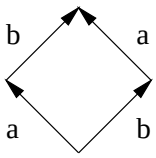


Fig. 5. Interleaving semantics of $a \mid b$

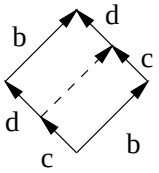


Fig. 6. A refinement of $a \mid b$

course, known long ago and was entirely graph-theoretic, and is known as the *request graph*. Figure 2 depicts the request graph corresponding to the progress graph of Figure 1. Nodes of this graph are resources of the concurrent system, that is, in this case, semaphores. There is an oriented edge from a resource x to a resource y if there is a process that has locked x and needs to lock y at a given time. A sufficient condition for such systems to be deadlock-free is that their corresponding request graphs be acyclic[†]. Unfortunately, this is not a necessary condition in general. For instance a request graph cannot capture the notion of n-semaphores, that is, resources that can be shared by up to n processes but not $n+1$ (for instance, asynchronous buffers of communication with size n that can be 'written' on by at most n processes). In fact, this really calls for some higher-dimensional versions of graphs.

Starting from progress graphs, the article Coffman *et al.* (1971) developed an algorithm in $O(n^2)$ (n is the number of tasks) to determine freedom from deadlocks. The notion of unsafe state was also introduced with the hope of determining automatically the correct schedulers that would prevent the whole system from running into a deadlock situation. However, this was limited to binary semaphores only[‡]. A fully worked out deadlock detection algorithm on progress graphs (including the determination of the unsafe region) that takes care of this limitation is described in Carson and Reynolds Jr. (1987). Unfortunately, this was not an optimal algorithm.

3. Semantics and analysis of concurrency

The semantics community came back to these geometric considerations with the development of 'truly-concurrent' semantics, as opposed to 'interleaving' semantics. The basis of the argument was that interleaving semantics, that is, the representation of parallelism by non-determinism ignores real asynchronous behaviours that actually exist[§]: $a \mid b$ where a and b are atomic is represented by the same transition system as the non-deterministic choice a then b or b then a (see Figure 5).

This fact creates problems, in particular, in the static analysis of (asynchronous) concurrent systems in that interleaving builds a lot of uninteresting states in the modelisation, and hence induces a high cost in verification. This is called the *state-space explosion problem*. Some techniques now exist for circumventing part of the problem, though they

[†] Note that this is a very geometric condition indeed.

[‡] There is a way to translate general semaphores into binary semaphores, see Dijkstra (1968), but this uses an encoding with integers that cannot be represented in progress graphs.

[§] For instance a distributed system that does not have a global clock is such a system.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

are actually mostly derived from truly-concurrent considerations (originally Petri nets in Valmari (1990) and Mazurkiewicz trace theory in Godefroid and Wolper (1991)). Another problem is that *refinement*, which is a very convenient technique in program analysis, is very difficult to apply in interleaving semantics, see van Glabbeek and Goltz (1989): for instance, suppose that action a is in fact non-atomic and is composed of two subactions c then d, then $a \mid b$ in interleaving semantics is no longer equivalent to a then b or b then a. The path c then b then d is missing there (see Figure 6, the missing part is represented by the dashed line). This implies that an interleaving semantics is bound to describe all atomic actions for the derived analyses to be correct, and hence incurring great complexity.

Quite a few models for true-concurrency have appeared (see, in particular, the account of Winskel and Nielsen (1994)) but it was only in 1991 that geometry was proposed as a means of solving the problem, in Pratt (1991). The diagnosis is that interleaving is only the boundary of the real picture. $a \mid b$ is really the filled-in square whose boundary is the non-deterministic choice a then b or b then a (the hollow square). The natural combinatorial notion, extension of transition systems, is that of a *cubical set*, which is a collection of points (states), edges (transitions), squares, cubes and hypercubes (higher-dimensional transitions representing the truly-concurrent execution of some number of actions). This was introduced in Pratt (1991) as well as possible formalizations using n-categories, and a notion of homotopy[†]. This is actually a combinatorial view of some kind of progress graph. Look back to Figure 1. Consider all interleavings of actions Pa, Pb, Va and Vb: they form a subgrid of the progress graph. Take as 2-transitions (that is, squares in the cubical set we are building) the filled-in squares. Only the forbidden region is really interleaved. Cubical sets generalize progress graphs, in that they allow any amount of non-deterministic choices as well as the dynamic creation of processes. These cubical sets are called *Higher-Dimensional Automata* (HDA) following Pratt (1991) because it really makes sense to consider a hypercube as some form of transition. Actually, at about the same time, a bisimulation semantics was given in van Glabbeek (1991). Notice that 2-transitions or squares are nothing but a *local commutation relation* as in Mazurkiewicz trace theory (Mazurkiewicz 1986), *independence relation* as in asynchronous transition systems (Bednarczyk 1988; Shields 1985), as in trace automata (as used in, for example, Kahn (1974) and Kahn and MacQueen (1977)), as in transition systems with independence (Sassone *et al.* 1994), or (indirectly) as with the 'confluence' relation of concurrent transition systems (Stark 1989). There are two more ingredients with HDA: the elegance and the power of the tools of geometric formalisations, and the natural generalisation to higher-dimensions (that is, 'higher-order independence relation' or n-ary independence relations).

Later semantic papers on the subject were very much influenced by Pratt (1991). In 1992, homological methods (see Mac Lane (1963) for an introduction) for studying the properties of Higher-Dimensional Automata were advocated in Goubault and Jensen (1992) given that they provide computable invariants of homotopy (at least in the usual case). A

[†] Which appeared later, see Fajstrup *et al.* (1999), not to be completely adequate.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

semantics of CCS was also discussed in a suitable category of HDA (also studied in Lanzmann (1993)) together with a notion of bisimulation[†]. This homology theory is not the definitive one: in particular, it cannot distinguish some situations that are quite simple. Some further work was needed and was begun in Gaucher (1997a) and Gaucher (1997b). In '*Homotopy invariants of higher-dimensional categories and concurrency in computer science*' in this issue, P. Gaucher extends his work in using strict globular ω-categories (generalizing Pratt (1991)) to formalize the execution paths of a parallel automaton. He also defines three new homology theories, which are more adequate than the one proposed in Goubault (1995a). The first properties of these theories are proved: in particular, Hurewicz morphisms, are constructed, relating homotopy and homology. It is to be noted that while this was being written up, R. Brown from Bangor University and R. Steiner from Glasgow University made an important contribution, which was very much related to this, by showing that the category of ω-categories is equivalent to the category of cubical sets with connections and compositions (see Brown and Higgins (1981a) and Brown and Higgins (1981b) for an introduction).

More general languages were modelled in Goubault (1993) with HDAs. Then a few analyses of programs by abstract interpretation (see Cousot and Cousot (1977) for an introduction) were designed: some earlier steps were given in Cridlig and Goubault (1993); then the automatic determination of a superset of possible schedules from the geometry of executions (that is, histories of resource usage) was given in Goubault (1995a) and Goubault (1995b); and some applications to model-checking were given in Cridlig (1995) (for shared-memory programming languages), and in Cridlig (1996) (on CML, that is, message-passing languages). A prototype Parallel Pascal Analyser has been implemented along these lines by R. Cridlig, see `http://www.dmi.ens.fr/~cridlig`.

In '*On the classification of dipaths in geometric models of concurrency*' in this issue, M. Raussen focusses on the determination of the dihomotopy classes in cubical complexes: deadlock detection and serializability (see next section) being only special cases. An algorithm is given for the 2-dimensional case and some ideas are given for solving the general case. In '*Loops, ditopology and deadlocks*' in this issue, L. Fajstrup gives a geometric model of a truly-concurrent PV system of n processes with finitely many (nested) loops and proves that it suffices to study a finite number of deloopings of each loop to determine the unsafe region and the deadlocks. Hence the previous algorithm on deadlock detection of Fajstrup *et al.* (1998) can be applied to these finitely many deloopings to give the precise unsafe area in concurrent systems with loops.

Some proposals have been made to use the scheduling information obtained from the HDA semantics to derive automatic parallelization algorithms. This has been fully treated for CCS in Takayama (1995) and Takayama (1996).

Most of the models used since V. Pratt's article were based on some form of cubical set. Some more recent work (Fajstrup and Raussen (1996) and Fajstrup *et al.* (1998) in particular) has introduced topological models, the *local po-space* models, generalizing the previous models, in order to reason in a similar way as in ordinary algebraic topology,

[†] There is also a way to fully compute the branchings, mergings and deadlocks inductively on this language.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

that is, by reasoning directly on 'continuous' shapes and not through their combinatorial representations (simplicial sets in general). Of course, as in the standard case, there is, as shown in Fajstrup *et al.* (1999) again, a natural relationship between combinatorial and topological representations through a pair of adjoint functors, geometric realization and singular cube functors. The expected applications of this modelization, as well as some of the basic notions about 'directed homotopy' and schedules are described in Fajstrup *et al.* (1999). More recently, V. Pratt proposed *Chu Spaces* (see for instance Pratt (1999)) as a model for concurrency, returning to the failure identified in Pratt (1991) of the natural duality event/schedule, in interleaving semantics. V. Pratt in '*Higher-dimensional automata revisited*' in this issue, develops this idea and expresses the HDA geometric model in terms of Chu spaces.

Some potentially related semantic models are the *n*-categorical formulations of Buckland and Johnson (1996) and some other recent combinatorial or categorical formulations in Fiore *et al.* (1997), Sassone and Cattani (1996), Sokolowski (1998a), Sokolowski (1998b) and Sokolowski (1998c).

4. Correctness of distributed databases

Another strand of research is concerned with distributed databases, and this is, in fact, very much related to the 'multiprogramming' and 'semaphore' strand described earlier. However, the kind of properties that have been studied are slightly different, and the focus has been put on devising optimal algorithms for the analysis of simple transaction models. As a matter of fact, a distributed database can be seen as a shared-memory machine (containing items) on which processes (called transactions) act by reading and writing, getting permissions to do so by using the appropriate functions on attached semaphores. One of the main purposes of this area is to ensure coherence of the distributed database while ensuring good performance, through a definition of suitable policies (protocols) for transactions to perform their own actions (with P and V). This means that deadlock-freedom of transactions is of importance. Correctness of a distributed database is itself very often expressed by some form of a *serializability* condition. For example, consider Figure 2. This could describe a database with two transactions $T_1 = Pb.Vb.Pa.Va$ and $T_2 = Pa.Va.Pb.Vb$ trying to modify two items a and b. All paths of execution above the left hole are equivalent to a serial execution of transaction T_2 then transaction T_1. All paths of execution below the right hole are equivalent to the serial execution of transaction T_1 then T_2. The third type of dipath is not a serial dipath – it describes several equivalent cases: for instance, T_1 acquires b, T_2 acquires a, then T_1 acquires a and T_2 acquires b. Think of the database as representing airplane tickets (for instance b is the return ticket corresponding to the one-way ticket a), and the two transactions as representing remote booking booths, the action between a P and its corresponding V is writing a name on the ticket. The situation here is that T_1 will have reserved its one-way ticket and T_2 will have reserved its return ticket only. This is not an allowed behaviour. It is not equivalent to a purely serial schedule, which are the only ones that are specified as correct (only one of T_1 or T_2 gets both tickets).

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

Testing serializability is unfortunately known to be an NP-complete problem (Papadimitriou 1979), even when the model is only based on simple binary semaphores.

The progress graph approach to the study of distributed databases was really initiated in Yannakakis *et al.* (1979) and then in Papadimitriou (1983). Lipski and Papadimitriou (1981) described an algorithm for proving the safety through serializability of distributed databases with only two transactions expressed as progress graphs. The underlying algorithmics relies on proving the connectedness of the closure of the set of forbidden rectangles†. Of course, the real problem in our previous example was that the forbidden region was disconnected, allowing dipaths to interleave some of the requests of different transactions. Another algorithm, for proving freedom from deadlock for two transactions synchronizing with binary semaphores only was also described. It was shown to have $O(n \log n \log \log n)$ (where n is the number of forbidden rectangles) time complexity. A notion of 'directed homotopy' was also defined. The generalization of safety conditions to higher-dimensions through the method of Yannakakis *et al.* (1979) to reduce it to 2-dimensional problems, was then shown to be of $O(nd2^d + d^2 \log n \log \log n)$ time complexity (d is the dimension, that is, the number of transactions). Much work has been done on the algorithmics of these geometric problems, and the algorithm above for safety was improved (actually it is made optimal) in Soisalon-Soininen and Wood (1985), achieving $O(n \log n)$ time and $O(n)$ space complexity for 2 transactions, and then relying on M. Yannakakis' result for the extension to any dimension. This is not the best that can be done in higher dimensions: the next step is achieved in Fajstrup *et al.* (1998) where a direct method for unsafe regions is described and where it is shown that the closure of the forbidden region in higher-dimension is not a strong enough condition for serializability in general. An application to proving the 2-phase locked protocol is given in Gunawardena (1994), and, using dihomotopy, in Fajstrup *et al.* (1999).

5. Computability and complexity of fault-tolerant distributed protocols

The strand of work here is concerned with the robust or fault-tolerant implementation of distributed programs. More precisely, the interest here is in *wait-free* or *t-resilient* ($0 \leqslant t \leqslant n-1$ where n is the number of processors involved, wait-free being $n-1$-resilient) implementations on a distributed machine composed of several units communicating through a shared memory via atomic read/write or FIFO queues *etc.*, or through synchronous/asynchronous message passing. This means that the processes executed on the n processors must be as loosely coupled as possible so that even if t processors fail to terminate, the others will carry on their computation and find a correct partial result (as observed on the non-faulty ones).

Consider, as an example, a machine with two processors, P_1 and P_2, communicating by atomic reads and writes on a shared memory. Each of these processes has a local binary variable x_1 (for P_1) and x_2 (for P_2), respectively. The binary *consensus* problem is to design an algorithm for P_1 and P_2 to agree on one value, which, moreover, has to be a value that

† This is also cited in the 'The geometry of rectangles' chapter in Preparata and Shamos (1993).

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

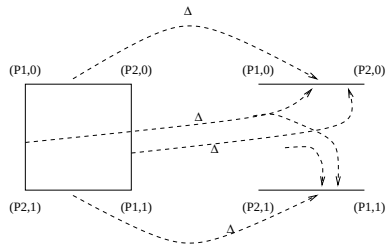


Fig. 7. The binary consensus decision map from initial global states to final global states

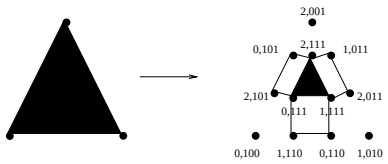


Fig. 8. The synchronous protocol complex (for 3 processes) after one round

one of them started with. Therefore, if $x_1 = 0$ and $x_2 = 1$ at the beginning, we want them to be $x_1 = 0$ and $x_2 = 0$ or $x_1 = 1$ and $x_2 = 1$ at the end. Of course, this is complicated by the fact that we ask the solution to be wait-free. If one of the two processes fails (the other process being unable to know whether this process is dead or just very slow), then the non-faulty one must terminate with one of the values P_1 or P_2 started with originally.

Unfortunately, it was proved in Fisher *et al.* (1985) that this is not possible – in fact it was proved for the equivalent case of an asynchronous message-passing system, in the general 1-resilient case. The article used an argument from graph theory (but was already quite geometric in nature). It in fact originated the geometric point of view on this field of research, as we will explain†.

The main argument can be understood in broad terms ('similarity chain') as a connectedness result. If we represent the local states of each processor P_i ($i = 1, 2$), at some point of the execution of a program, by a vertex (P_i, x_i) and the global states of the system by an (un-oriented) edge linking the two local states that compose them, then it is easy to see that the semantics of atomic reads and writes requires that the reachable global states starting from one initial global state have to form a connected graph. There is no way a program on such a machine can 'break' connectivity. This implies that the binary consensus cannot be implemented here since we must be able to reach from the global state $((P_1, 0), (P_2, 1))$ two disconnected global states $((P_1, 0), (P_2, 0))$ and $((P_1, 1), (P_2, 1))$, as shown in Figure 7.

Further work generalizes this simple geometric argument to the required higher-dimensional cases. Biran *et al.* (1988) gave a characterization of a class of problems solvable in asynchronous message-passing systems in the presence of a single failure. No generalization to more failures has since been solved using the same kind of graph techniques. It was then a generally shared belief that one would have to use more powerful techniques in that case. The conjecture in Chaudhuri (1990) that the k-set agreement problem (a kind of weak consensus; all is required is that the non-faulty processes eventually agree on a subset of at most k values taken from the input values) cannot be solved in certain

† We refer the reader to the book Lynch (1996) for a flavour of the vast number of results that have been proved in the field of fault-tolerant distributed protocols.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

asynchronous systems was finally proved independently in Borowsky and Gafni (1993), Saks and Zaharoglou (1993), and Herlihy and Shavit (1993).

The basic idea of Herlihy and Shavit (1993) is to generalize such pictures as the one of Figure 7 using *simplicial sets* instead of graphs (which are special cases of the former). Simplicial sets are made up of vertices and edges, but also triangles and simplexes in general glued together. A simplex of dimension n represents the global state, at some point of the execution, of n processes. Vertices are still pairs composed of the name of the process, together with its local state. Again, given an initial global state, the semantics of the operations of the distributed machine we want to study is defined by the reachable global states at any time. For instance, Figure 8 shows the simplicial set, called the protocol complex, after one round of communication on a synchronous message-passing machine, which broadcasts the local states to all processes at each step. If there is a simplicial map from it to some suitable set of global states, respecting the specification of the problem, then there exists a corresponding wait-free protocol. This enables us also to compute how many rounds of communication might be necessary to solve a given problem. Notice that the simplexes of the protocol complex are really the schedules, and this should be related to the directed homotopy approach of Section 3. This has already been hinted at in Goubault (1996a), Goubault (1996b) and Goubault (1997), but for simple cases only.

It was also proved in Herlihy and Shavit (1993) that in a shared-memory model with single reader/single writer registers providing atomic read and write operations, k-set agreement requires at least $\lfloor f/k \rfloor + 1$ rounds where f is the number of processes that can fail.

The renaming task (processes must try to agree on a smaller set of names between each other than the original set of names), first proposed in Attiya *et al.* (1990) was also finally solved in Herlihy and Shavit (1993). There is a wait-free protocol for the renaming task in certain asynchronous systems if the output name space is sufficiently large. It was already known that there is a wait-free solution for the renaming task for $2n+1$ or more output names on a system of $n+1$ asynchronous processors and none for $n+2$ or fewer output names. M. Herlihy and N. Shavit refined this result and showed that there was no solution for strictly less that $2n+1$ output names.

As well as giving such impossibility results, this work can also provide constructive means for finding algorithms (see, for instance, Herlihy and Shavit (1994)).

More generally, datatypes do matter for computability results. It has been known for quite a long time that consensus for two processes can be solved with shared memory with atomic reads and writes plus a FIFO queue, or with the addition of an atomic test&set operation. If we define the consensus number of a data type as the maximal number of asynchronous processors (having atomic read and write) on which it can implement wait-free consensus, then,

— atomic read/write registers have consensus number 1,
— test&set and fetch&add registers, queues, and stacks have consensus number 2,
— n-register assignment has consensus number $2n-2$,
— load-locked, store-conditional and compare-and-swap registers have consensus number ∞.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

These facts motivated the introduction of the following general problem, dealing with the power of the architecture of distributed machines. We say that a datatype, or object, is an (m, j)-consensus object if it allows any set of m processes to solve j-set agreement tasks. Herlihy and Rajsbaum proved in Herlihy and Rajsbaum (1994) that it is impossible to implement $(n+1, k)$-consensus using (m, j)-consensus objects if $n/k > m/j$ (see also Borowsky and Gafni (1993)).

Further work on this can be found in Jayanti (1993), Jayanti (1997) and Schenk (1997). In particular, the problem identified in Jayanti (1997) is that the hierarchy of data objects as briefly sketched above is not robust, in the sense that it is possible to implement some datatypes with consensus number k using several datatypes with consensus numbers strictly less than k.

This, of course, is due to some subtle interactions between the use of these data objects, and we could hope that the more general study of the schedules with such datatypes might lead to some better classifications.

In '*Algebraic spans*' in this issue, M. Herlihy and S. Rajsbaum introduce a new tool (related to the one described in Herlihy and Rajsbaum (1995)) for proving impossibility results, based on a core theorem of algebraic topology, the acyclic carrier theorem, which unifies, generalizes and extends earlier results.

6. Some perspectives

There are numerous perspectives in the static analysis of concurrent programs, and the computability and complexity issues in fault-tolerant distributed systems, as well as in concurrent database theory, as I have tried to explain in the previous sections. The aim here is not to list the possible research that could be carried on in these directions (good references for this are Fajstrup *et al.* (1999) and Herlihy and Rajsbaum (1999)), but to look at other possible uses of these techniques. For instance, Squier's theorem in rewriting systems theory, which gives a necessary condition for the existence of a presentation of a given monoid by a finite canonical rewriting system in terms of its homology (it must be of finite dimension), seems very much related to the techniques presented above. It is definitely a computability result, as we have in fault-tolerant distributed systems theory, but for something that looks sequential (rewriting). As hinted at in Goubault (1995a), this can be understood as a problem of concurrency theory in that the study of the confluence of rewriting systems is related to parallel reduction techniques (as in Lévy (1978) for instance). The resolutions used in most of the proofs of this theorem, as in Kobayashi (1990), Groves (1991), Anick (1986), Farkas (1992) and Lafont and Prouté (1990) are very much like a Knuth–Bendix completion procedure, where higher-dimensional objects are filling in possible defects of local confluence. This looks like building higher-dimensional transitions implementing the parallel (confluent) reductions (see, in particular, Groves (1991), where the resolution is a cubical complex, and in dimension one it is generated by the transition system coming from the reduction relation). Some other proof techniques use something that is very much like some kind of directed homotopy, as in Squier *et al.* (1994), for instance. Other interesting relations should be studied in the area of 'higher-dimensional' word problems, as in Burroni (1991).

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

A hope is that geometry can also give some insight in logics, especially modal logics as in Goubault-Larrecq and Goubault (1999). Finally, there is some intuition from theoretical physics that seems relevant to semantics, in particular concerning time and dynamical systems. Some of the concepts of M. Raussen's and L. Fasjtrup's articles in this issue are based on similar notions to those in Penrose (1972): some areas of physics (but not classical mechanics) have to consider time as non-reversible, and hence have to construct some kind of directed topology.

References

Anick, D. J. (1986) On the homology of associative algebras. *Transactions of the American Mathematical Society* **296** 641–659.

Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D. and Reischuk, R. (1990) Renaming in an asynchronous environment. *Journal of the ACM* **37** (3) 524–548.

Baues, H. J. (1989) *Algebraic homotopy*. Cambridge Studies in Advanced Mathematics **15**, Cambridge University Press.

Bednarczyk, M. A. (1988) *Categories of asynchronous systems*, Ph. D. thesis, University of Sussex.

Biran, O., Moran, S. and Zaks, S. (1988) A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In: *Proc. 7th Annual ACM Symposium on Principles of Distributed Computing*, ACM Press 263–275.

Borowsky, E. and Gafni, E. (1993) Generalized FLP impossibility result for t-resilient asynchronous computations. In: *Proc. of the 25th STOC*, ACM Press.

Brown, R. and Higgins, P. J. (1981a) Colimit theorems for relative homotopy groups. *Journal of Pure and Applied Algebra* **22** 11–41.

Brown, R. and Higgins, P. J. (1981b) On the algebra of cubes. *Journal of Pure and Applied Algebra* **21** 233–260.

Buckland, R. and Johnson, M. (1996) ECHIDNA: A system for manipulating explicit choice higher dimensional automata. In: *AMAST'96: Fifth Int. Conf. on Algebraic Methodology and Software Technology*, Munich.

Burroni, A. (1991) Higher dimensional word problem. *Springer-Verlag Lecture Notes in Computer Science* **530**.

Carson, S. D. and Reynolds Jr, P. F. (1987) The geometry of semaphore programs. *ACM Transactions on Programming Languages and Systems* **9** (1) 25–53.

Chaudhuri, S. (1990) Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In: *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, ACM Press 311–334.

Coffman, E. G., Elphick, M. J. and Shoshani, A. (1971) System deadlocks. *Computing Surveys* **3** (2) 67–78.

Cousot, P. and Cousot, R. (1977) Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points. *Principles of Programming Languages* **4** 238–252.

Cridlig, R. (1995) Semantic analysis of shared-memory concurrent languages using abstract model-checking. In: *Proc. of PEPM'95, La Jolla*, ACM Press.

Cridlig, R. (1996) Semantic analysis of Concurrent ML by abstract model-checking. In: *Proceedings of the LOMAPS Workshop*.

Cridlig, R. and Goubault, E. (1993) Semantics and analyses of Linda-based languages. In: Proc. of WSA'93. *Springer-Verlag Lecture Notes in Computer Science* **724**.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

Dijkstra, E. (1968) *Cooperating Sequential Processes*, Academic Press.

Fajstrup, L., Goubault, E. and Raussen, M. (1998) Detecting deadlocks in concurrent systems. In: *Proceedings of the 9th International Conference on Concurrency Theory*, Springer-Verlag (also available at `http://www.dmi.ens.fr/~goubault`).

Fajstrup, L., Goubault, E. and Raussen, M. (1999) Algebraic topology and concurrency. *Theoretical Computer Science* (submitted) (also technical report, Aalborg University).

Fajstrup, L. and Raussen, M. (1996) Detecting Deadlocks in Concurrent Systems. Technical report, BRICS Research Report, Aalborg University.

Farkas, D. R. (1992) The Anick resolution. *Journal of Pure and Applied Algebra* **79**.

Fiore, M., Plotkin, G. and Power, J. (1997) Complete cuboidal sets in axiomatic domain theory (extended abstract). In: *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland*, IEEE Computer Society Press 268–279.

Fisher, M., Lynch, N. A. and Paterson, M. S. (1985) Impossibility of distributed commit with one faulty process. *Journal of the ACM* **32** (2) 374–382.

Gabriel, P. and Zisman, M. (1967) Calculus of fractions and homotopy theory. In: *Ergebnisse der Mathematik und ihrer Grenzgebiete* **35**, Springer-Verlag.

Gaucher, P. (1997a) Connexion de flux d'information en algèbre homologique. Technical report, IRMA, Strasbourg, available at `http://irmasrv1.u-strasbg.fr/~gaucher`.

Gaucher, P. (1997b) Etude homologique des chemins de dimension 1 d'un automate. Technical report, IRMA, Strasbourg, available at `http://irmasrv1.u-strasbg.fr/~gaucher`.

Godefroid, P. and Wolper, P. (1991) Using partial orders for the efficient verification of deadlock freedom and safety properties. In: Proc. of the Third Workshop on Computer Aided Verification. *Springer-Verlag Lecture Notes in Computer Science* **575** 417–428.

Goubault, E. (1993) Domains of higher-dimensional automata. In: *Proc. of CONCUR'93, Hildesheim*, Springer-Verlag.

Goubault, E. (1995a) *The Geometry of Concurrency*, Ph. D. thesis, Ecole Normale Supérieure (also available at `http://www.dmi.ens.fr/~goubault`).

Goubault, E. (1995b) Schedulers as abstract interpretations of HDA. In: *Proc. of PEPM'95, La Jolla*, ACM Press (also available at `http://www.dmi.ens.fr/~goubault`).

Goubault, E. (1996a) The Dynamics of Wait-Free Distributed Computations. Technical report, Research Report LIENS-96-26.

Goubault, E. (1996b) A semantic view on distributed computability and complexity. In :*Proceedings of the 3rd Theory and Formal Methods Section Workshop*, Imperial College Press (also available at `http://www.dmi.ens.fr/~goubault`).

Goubault, E. (1997) Optimal implementation of wait-free binary relations. In: *Proceedings of the 22nd CAAP*, Springer-Verlag.

Goubault, E. and Jensen, T. P. (1992) Homology of higher-dimensional automata. In: *Proc. of CONCUR'92, Stonybrook, New York*, Springer-Verlag.

Goubault-Larrecq, J. and Goubault, E. (1999) Order-theoretic, geometric and combinatorial models of intuitionistic s4 proofs. In: *proceedings of IMLA'99.*

Groves, J. R. J. (1991) Rewriting systems and homology of groups. In: Kovacs, L. G. (Ed.) Groups – Canberra 1989. *Springer-Verlag Lecture Notes in Mathematics* **1456** 114–141.

Gunawardena, J. (1994) Homotopy and concurrency. In: *Bulletin of the EATCS* **54** 184–193.

Herlihy, M. and Rajsbaum, S. (1994) Set consensus using arbitrary objects. In: *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing*, ACM Press.

Herlihy, M. and Rajsbaum, S. (1995) Algebraic spans (preliminary version). In: *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada* 90–99.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

Herlihy, M. and Rajsbaum, S. (1999) New perspectives in distributed computing. In: Kutylowski, M., Pacholski, L. and Wierzbicki, T. (eds.) 24th International Symposium on Mathematical Foundations of Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **1672** 170–186.

Herlihy, M. and Shavit, N. (1993) The asynchronous computability theorem for t-resilient tasks. In: *Proc. of the 25th STOC*, ACM Press.

Herlihy, M. and Shavit, N. (1994) A simple constructive computability theorem for wait-free computation. In: *Proceedings of STOC'94*, ACM Press.

Jayanti, P. (1993) On the robustness of Herlihy's hierarchy. In: *Proceedings of the Twelth Annual ACM Symposium on Principles of Distributed Computing, Ithaca, New York* 145–157.

Jayanti, P. (1997) Robust wait-free hierarchies. *Journal of the ACM* **44** (4) 592–614.

Kahn, G. (1974) The semantics of a simple language for parallel programming. *Information Processing* **74**.

Kahn, G. and MacQueen, D. B. (1977) Coroutines and networks of parallel processes. *Information Processing* **77**.

Kobayashi, Y. (1990) Complete rewriting systems and homology of monoid algebras. *Journal of Pure and Applied Algebra* **65** 263–275.

Lafont, Y. and Prouté, A. (1990) Church–Rosser Property and Homology of Monoids. Technical report, Ecole Normale Supérieure.

Lanzmann, E. (1993) Automates d'ordre supérieur, Master's thesis, Université d'Orsay.

Lévy, J.-J. (1978) *Réductions Correctes et Optimales dans le Lambda-Calcul*, Ph. D. thesis, Université Paris VII.

Lipski and Papadimitriou (1981) A fast algorithm for testing for safety and detecting deadlocks in locked transaction. *ALGORITHMS: Journal of Algorithms.*

Lynch, N. (1996) *Distributed Algorithms*, Morgan-Kaufmann.

Mac Lane, S. (1963) Homology. In: *Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen* **114**, Springer Verlag.

May, J. P. (1967) *Simplicial objects in algebraic topology*, van Nostrand.

Mazurkiewicz, A. (1986) Trace theory. In: Rozenberg, G. (ed.) Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, PART II, PO of an Advanced Course. *Springer-Verlag Lecture Notes in Computer Science* **255** 279–324.

Papadimitriou, C. H. (1979) The serializability of concurrent database updates. *Journal of the ACM* **26** (4) 631–653.

Papadimitriou, C. H. (1983) Concurrency control by locking. *SIAM Journal on Computing* **12** (2) 215–226.

Penrose, R. (1972) Techniques of Differential Topology in Relativity. *Conference Board of the Mathematical Sciences, Regional Conference Series in Applied Mathematics* **7**, SIAM.

Pratt, V. (1991) Modeling concurrency with geometry. In: *Proc. of the 18th ACM Symposium on Principles of Programming Languages*, ACM Press.

Pratt, V. (1999) Chu spaces. In: Course notes for the School in Category Theory and Applications, Coimbra, Portugal.

Preparata, F. P. and Shamos, M. I. (1993) *Computational Geometry, an Introduction*, Springer-Verlag.

Saks, M. and Zaharoglou, F. (1993) Wait-free k-set agreement is impossible: The topology of public knowledge. In: *Proc. of the 25th STOC*, ACM Press.

Sassone, V. and Cattani, G. L. (1996) Higher-dimensional transition systems. In: *Proceedings of LICS'96.*

Sassone, V., Nielsen, M. and Winskel, G. (1994) Relationships between models of concurrency. In: *Proceedings of the Rex'93 school and symposium.*

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE

Schenk, E. (1997) The consensus hierarchy is not robust. In: *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California* 279.

Shields, M. (1985) Concurrent machines. *Computer Journal* **28**.

Shoshani, A. and Coffman, E. G. (1970) Sequencing tasks in multiprocess systems to avoid deadlocks. In: *Conference Record of 1970 Eleventh Annual Symposium on Switching and Automata Theory, Santa Monica, California*, IEEE 225–235.

Soisalon-Soininen, E. and Wood, D. (1985) An optimal algorithm for testing for safety and detecting deadlocks in locked transaction systems. In: *Symposium on Principles of Database Systems (PODS '82)* 108-116.

Sokolowski, S. (1998a) Homotopy in concurrent processes. Technical report, Institute of Computer Science, Gdansk Division.

Sokolowski, S. (1998b) Investigation of concurrent processes by means of homotopy functors. Technical report, Institute of Computer Science, Gdansk Division.

Sokolowski, S. (1998c) Point glueing in cpo-s. Technical report, Institute of Computer Science, Gdansk Division.

Spanier, E. J. (1966) *Algebraic Topology*, McGraw Hill.

Squier, C. C., Otto, F. and Kobayashi, Y. (1994) A finiteness condition for rewriting systems. *Theoretical Computer Science* **131** 271–294.

Stark, A. (1989) Concurrent transition systems. *Theoretical Computer Science* **64** 221–269.

Takayama, Y. (1995) Cycle filling as parallelization with expansion law (submitted).

Takayama, Y. (1996) Extraction of concurrent processes from higher-dimensional automata. In: *Proceedings of CAAP'96* 72-85.

Valmari, A. (1990) A stubborn attack on state explosion. In: Proc. of CAV'90. *Springer-Verlag Lecture Notes in Computer Science*.

van Glabbeek, R. (1991) Bisimulation Semantics for Higher Dimensional Automata. Technical report, Stanford University (also available at `http://theory.stanford.edu/~rvg/had`).

van Glabbeek, R. and Goltz, U. (1989) Partial order semantics for refinement of actions. *Bulletin of the EATCS* **34**.

Winskel, G. and Nielsen, M. (1994) Models for concurrency. *Handbook of Logic in Computer Science* **3**, Oxford University Press.

Yannakakis, M., Papadimitriou, C. H. and Kung, H. T. (1979) Locking policies: Safety and freedom from deadlock. In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, IEEE 286–297.

Downloaded from https://www.cambridge.org/core. Columbia University Libraries, on 05 Jul 2017 at 19:40:35, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://www.cambridge.org/core/product/513EC7F7270A2AAD1F0833F9A713E7AE