

From Geometric Semantics to Asynchronous Computability

Éric Goubault¹, Samuel Mimram¹, and Christine Tasson²

¹ LIX, École Polytechnique, Palaiseau, France

² PPS, Université Paris 7, France

Abstract. We show that the protocol complex formalization of fault-tolerant protocols can be directly derived from a suitable semantics of the underlying synchronization and communication primitives, based on a geometrization of the state space. By constructing a one-to-one relationship between simplices of the protocol complex and dihomotopy classes of dipaths in the latter semantics, we describe a connection between these two geometric approaches : protocol complexes and directed algebraic topology. This is exemplified on atomic snapshot, iterated snapshot and layered immediate snapshot protocols, where a well-known combinatorial structure, interval orders, plays a key role. We believe that this correspondence between models will extend to proving impossibility results for much more intricate fault-tolerant distributed architectures.

1 Introduction

Fault-tolerant distributed computing is concerned with designing algorithms, and, when possible, solving so-called *decision tasks* on a given distributed architecture, in the presence of faults. The seminal result in this field was established by Fisher, Lynch and Paterson in 1985, who proved the existence of a simple task that cannot be solved in a message-passing system (or in shared memory [27]) with at most one potential crash [11]. In particular, there is no way in such a distributed system to solve the very fundamental consensus problem: each processor starts with an initial value in local memory, typically an integer, and should end up with a common value, which is one of the initial values.

Later on, Biran, Moran and Zaks developed a characterization of the decision tasks that can be solved by a (simple) message-passing system in the presence of one failure [3]. The argument uses a “similarity chain”, which can be seen as a connectedness result of a representation of the space of all reachable states, called the *view complex* [25] or the *protocol complex* [24]. Of course, this argument turned out to be difficult to extend to models with more failures, as higher-connectedness properties of the protocol complex matter in these cases. This technical difficulty was first tackled, using homological considerations, by Herlihy and Shavit [23] (and independently [5,31]): there are simple decision tasks, such as k -set agreement, a weaker form of consensus, that cannot be solved for $k < n$ in the wait-free asynchronous model, i.e. shared-memory distributed protocols on n processors, with up to $n - 1$ crash failures. Then, the full characterization of wait-free asynchronous decision tasks with atomic reads and writes (or equivalently, with atomic snapshots) was described by Herlihy and Shavit [24]: this relies on the central notion of chromatic (or colored) simplicial complexes, and their

subdivisions. All these results stem from the contractibility of the “standard” chromatic subdivision, which was completely formalized in [25,26] (and even for *iterated* models [19]) and corresponds to the *protocol complex* of distributed algorithms solving layered immediate snapshot protocols.

Over the years, the geometric approach to problems in fault-tolerant distributed computing has been very successful, see [22] for a fairly complete up-to-date treatment. One potential limitation however is that for some intricate models, it is extremely difficult to produce their corresponding protocol complex. In this paper, we are exploring the links between the semantics of the synchronization and communication primitives we are considering on a given distributed architecture, and the protocol complex. The interest is that the semantics of such synchronization primitives is much simpler to write down than the protocol complex, which is very error-prone to describe, as we will see in Section 3.2. We advocate in this paper the calculation of protocol complexes directly from the formal semantics of the underlying synchronization primitives.

The other aim of this article is to make the link between two geometric theories of concurrent and distributed computations: one based on protocols complexes, and the other, based on *directed algebraic topology*. Actually, the semantics of concurrent and distributed systems can be given by topological models, as pushed forward in a series of seminal papers in concurrency, in the early 1990s. These papers have explored the use of precubical sets and *Higher-Dimensional Automata* (which are labeled precubical sets equipped with a distinguished beginning vertex) [30,32], begun to suggest possible homology theories [17,18,6] and pushed the development of a specific homotopy theory, part of a general *directed algebraic topology* [20]. On the practical side, directed topological models have found applications to deadlock and unreachable state detection [9], validation and static analysis [15,4,7], state-space reduction (as in e.g. model-checking) [16], serializability and correctness of databases [21] (see also [14,10] for a panorama of applications).

In order to instantiate this link, we will be considering the simple model of shared-memory concurrent machines with crash failures, where processors compute and communicate through shared locations, and where reads and writes are supposed to be atomic. This model can also be presented [28] as *atomic snapshot protocols* [1,2], where processors are executing the following instructions: scanning the entire shared memory (and copying it into their local memory), computing in its local memory, and updating its “own value”, i.e. writing the outcome of its computation in a specific location in global memory, assigned to him only. The methodology we are describing here is by no means limited to this simple model: we have provided in this paper a general framework that builds protocol complexes from the semantics of communication primitives. However, what is more difficult is determining the set of *directed homotopy classes* of directed paths in this semantics. This is one of the reasons why we chose to exemplify the method on a well-known and simple case in fault-tolerant distributed computing. In general, this step is by no means trivial, reinforcing the need for formally deriving protocol complexes from semantics. The other reason is that the reader will be more familiar with the model and the expected result, and will be able to focus on the new technical (directed algebraic topological) aspects of the paper.

Contents of the paper and main contributions. Section 2.1 begins by defining the *standard semantics* (or interleaving semantics) of atomic read/write protocols, and more precisely of atomic snapshot protocols where read and write primitives are replaced by update and (global) scan ones. In Section 2.2, we give an alternative *geometric semantics*, which encodes also independence of actions, as a form of *homotopy* in a geometric model. The very basics of *directed algebraic topology* have been introduced for this purpose, but we refer the reader to [20,8] for more details. Yet, for the wider picture, we prove the fact that (directed) homotopy encodes commutation of actions, in the form of an equivalence between the standard semantics and the geometric semantics. It is shown in Section 2.3, Proposition 4 that two traces in the interleaving semantics modulo commutation of actions induce dihomotopic (directed) paths in the geometric model. The converse is shown in Section 2.3, Proposition 9, using the combinatorial notion of *interval order* [12]. We then combine these results with the semantic equivalence of Proposition 6, Section 2.3; this is the first main contribution of the paper.

In Section 3, we turn to the other geometric model of distributed systems: protocol complexes. The second main contribution of the paper is developed in Section 3.2: the protocol complex for atomic snapshot protocols (possibly iterated) is derived from the geometric semantics of Section 2.2, through interval orders. We specify this construction in Section 3.3 to the case of *layered immediate snapshot* which is generally studied by most authors, since it is much simpler to study, and is enough to prove the classical impossibility theorems, as e.g. [23]. Our explicit description of the protocol complex in the latter case is the same as the one of [19] (linked as well to the equivalent presentation of [25]), see Theorem 19. Combined with the result of [19] it proves that the layered immediate snapshot protocols produce collapsible protocol complexes, for any number of rounds. It then implies the asynchronous computability theorem of [23] all the way from the semantics of the communication primitives.

2 Concurrent semantics of asynchronous read/write protocols

2.1 Interleaving semantics of atomic read/write protocols

In *atomic snapshot* protocols, n processes communicate through shared memory using two primitives: `update` and `scan`. Informally, the shared memory is partitioned in n parts, each one corresponding to one of the n processes. The part of the memory associated with process P_i , with $i \in \{0, \dots, n-1\}$, is the one on which process P_i can write, by calling `update`. This primitive writes onto that part of memory, a value computed from the value stored in a local register of P_i . Note that as the memory is partitioned, there are never any write conflicts on memory. Conversely, all processes can read the entire memory through the `scan` primitive. Note also that there are never any read conflicts on memory. Still, it is well known that atomic snapshot protocols are equivalent [28] with respect to their expressiveness in terms of fault-tolerant decision tasks they can solve, to the protocols based on atomic registers with atomic reads and writes. Generic snapshot protocols are such that all processes loop, any number of times, on the three successive actions: locally compute a *decision value*, `update` then `scan`. It is also known [23,24] that, as far as fault-tolerant properties are concerned,

an equivalent model of computation can be considered: the full-information protocol where, for each process, decisions are only taken at the end of the protocol, i.e. after rounds of `update` then `scan`, only remembering the history of communications.

Interleaving semantics and trace equivalence. Formally, we consider a fixed set \mathcal{V} of *values*, together with two distinguished subsets \mathcal{I} and \mathcal{O} of *input* and *output values*, the elements of $\mathcal{V} \setminus (\mathcal{I} \cup \mathcal{O})$ being called *intermediate values*, and an element $\perp \in \mathcal{I} \cap \mathcal{O}$ standing for an *unknown value*. We suppose that the sets of values and intermediate values are infinite countable, so that pairs $\langle x, y \rangle$ of values $x, y \in \mathcal{V}$ can be encoded as intermediate values, and similarly for tuples. We suppose fixed a number $n \in \mathbb{N}$ of processes. We also write $[n]$ as a shortcut for the set $\{0, \dots, n-1\}$, and \mathcal{V}^n for the set of n -tuples of elements of \mathcal{V} , whose elements are called *memories*. Given $v \in \mathcal{V}^n$ and $i \in [n]$, we write v_i for the i -th component of v . We write \perp^n for the memory l such that $l_i = \perp$ for any $i \in [n]$.

There are two families of memories, each one containing one memory cell for each process P_i : the *local memories* $l = (l_i)_{i \in [n]} \in \mathcal{V}^n$, and the *global (shared) memory*: $m = (m_i)_{i \in [n]} \in \mathcal{V}^n$. A *state* of a program is a pair $(l, m) \in \mathcal{V}^n \times \mathcal{V}^n$ of such memories. Processes can communicate by performing actions which consist in updating and scanning the global memory, using their local memory: we denote by u_i any `update` by the i -th process and s_i any of its `scan`. We write $\mathcal{A}_i = \{u_i, s_i\}$ and $\mathcal{A} = \bigcup_{i \in [n]} \mathcal{A}_i$ for the set of *actions*.

Formally, the effect of the actions on the state is defined by a *protocol* π which consists of two families of functions $\pi_{u_i} : \mathcal{V} \rightarrow \mathcal{V}$ and $\pi_{s_i} : \mathcal{V} \times \mathcal{V}^n \rightarrow \mathcal{V}$ indexed by $i \in [n]$ such that $\pi_{s_i}(x, m) = x$ for $x \in \mathcal{O}$. Starting from a state (l, m) , the effect of actions is as follows: u_i means “replace the contents of m_i by $\pi_{u_i}(l_i)$ ”, and s_i means “replace the contents of l_i by $\pi_{s_i}(l_i, m)$ ”.

A protocol is *full-information* when $\pi_{u_i}(x) = x$ for every $x \in \mathcal{V}$, i.e. each process fully discloses its local state in the global memory. A sequence of actions $T \in \mathcal{A}^*$ is called an *interleaving trace*, and we write $\llbracket T \rrbracket_\pi(l, m)$ for the state reached by the protocol π after executing the actions in T , starting from the state (l, m) . A sequence of actions $T \in \mathcal{A}^*$ is *well-bracketed* or *well-formed* (giving some form of generic protocol) when for every $i \in [n]$ we have $\text{proj}_i(T) \in (u_i s_i)^*$, where $\text{proj}_i : \mathcal{A}^* \rightarrow \mathcal{A}_i^*$ is the obvious projection which only keeps the letters in \mathcal{A}_i in a word over \mathcal{A} . We denote by \mathcal{A}^ω the set of countably infinite sequences of actions; such a sequence is well-bracketed when every finite prefix is.

It can be noticed that different interleaving traces may induce the same final local view for any process. Indeed, if $i \neq j$, then u_i and u_j modify different parts of the global memory, as we already noted informally, and thus $u_i u_j$ and $u_j u_i$ induce the same action on a given state. Similarly, s_i and s_j change different parts of the local memory, and thus $s_i s_j$ and $s_j s_i$ induce the same action on a given state. On the contrary, $u_i s_j$ and $s_j u_i$ may induce different traces as u_i may modify the global memory that is scanned by s_j . We thus define an *equivalence* \approx on interleaving traces, as the smallest congruence such that $u_j u_i \approx u_i u_j$ and $s_j s_i \approx s_i s_j$ for every indices i and j . Therefore :

Proposition 1. *The equivalence \approx of traces induces an operational equivalence: two equivalent interleaving traces starting from the same initial state lead to the same final state.*

This justifies that we consider traces *up to equivalence* in the following. We use the usual notions on such operational semantics: *execution traces*, *interleaving traces* will denote any finite sequences of actions u_i and s_i in \mathcal{A}^* , *maximal* execution traces are traces that cannot be further extended. We also use the classical notions of *length* and *concatenation* of execution traces.

Decision tasks. We are going to consider the possibility of solving a particular task with an asynchronous protocol. Formally, those tasks are specified as follows:

Definition 2. *A wait-free task specification Θ is a relation $\Theta \subseteq \mathcal{I}^n \times \mathcal{O}^n$ such that for all $(l, l') \in \Theta$, $i \in [n]$ s.t. $l_i = \perp$, and $x \in \mathcal{I}$, we also have $(l_i^x, l') \in \Theta$ where l_i^x is the memory obtained from l by replacing the i -th value by x . We note $\text{dom } \Theta = \{l \in \mathcal{I}^n \mid \exists l' \in \mathcal{O}^n, (l, l') \in \Theta\}$ for the domain of a wait-free task specification Θ and $\text{codom } \Theta = \{l' \in \mathcal{O}^n \mid \exists l \in \mathcal{I}^n, (l, l') \in \Theta\}$ for its codomain.*

Notice that $\text{dom } \Theta$ induces a simplicial complex, with $[n] \times (\mathcal{I} \setminus \{\perp\})$ as vertices, and simplices are of the form $\{(i, x) \in [n] \times \mathcal{V} \mid l_i = x \neq \perp\}$, for any $l \in \text{dom } \Theta$. This simplicial complex is called the *input complex*; the *output complex* is defined similarly from $\text{codom } \Theta$. We say that a protocol π *solves* a task specification Θ when for every $l \in \text{dom } \Theta$, and well-bracketed infinite sequence of actions $T \in \mathcal{A}^\omega$, there exists a finite prefix T' of T such that $(l, l') \in \Theta$ where l' is the local memory after executing T' , i.e. $(l', m') = \llbracket T' \rrbracket_\pi(l, \perp^n)$. It can be shown [24] that, w.r.t. task solvability, we can assume that $\text{dom } \Theta$ contains only the memory l such that $l_i = i$, for all i , and its faces; for simplicity we will do so in Section 3.

Of particular interest is the *view protocol* (sometimes identified with the full-information protocol in the literature) π^{\triangleleft} such that $\pi_{u_i}^{\triangleleft}(x) = x$ for $x \in \mathcal{V}$, i.e. the protocol is full-information, and $\pi_{s_i}^{\triangleleft}(x, m) = \langle x, \langle m \rangle \rangle$ for $x \in \mathcal{V}$ and $m \in \mathcal{V}^n$: when reading the global memory, the protocol stores (an encoding of) the pair constituted of its current local memory x and (an encoding as a value of) the global memory m it has read. This is akin to the use of *generic protocols in normal form* [24], where protocols only exchange their full history of communication for a fixed given number of rounds, and then apply a local decision function. It can be shown that the view protocol is the “most general one” (i.e. initial in a suitable category). Thus, we will be satisfied with describing the potential sets of histories of communication between processes, without having to encode the decision values: this is the basis of the geometric semantics of Section 2.2. As a direct consequence, we recover the usual definition of the solvability of a task as a simplicial map from some iterated protocol complex to the output complex [24,22].

2.2 Directed geometric semantics

In this section, we give an alternative semantics to atomic snapshot protocols, using a geometric encoding of the state space, together with a notion of “time direction”. One of

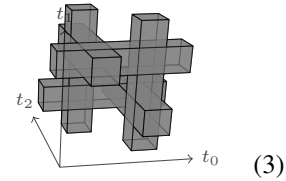
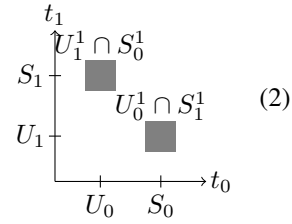
the most simple settings in which this can be performed is the one of pospaces [29,13]: a *pospace* is a topological space \mathbb{X} endowed with a partial order \leq such that the graph of the partial order is closed in $\mathbb{X} \times \mathbb{X}$ with the product topology. The intuition is that, given two points $x, y \in \mathbb{X}$ such that $x \leq y$, y cannot be reached before x . The encoding, or semantics of a concurrent or distributed protocol in terms of directed topological spaces of some sort can be done in a more general manner [7,8]. Here, we simply define, directly, the pospace that gives the semantics we are looking for. It is rather intuitive and we will check this is correct with respect to the interleaving semantics, in Section 2.3.

Consider the pospace $\mathbb{X}_{(r)}^n$ below, indexed by the number n of processes and the vector of number of rounds $(r) = (r_0, \dots, r_{n-1})$ (each $r_i \in \mathbb{N}$, with $i \in [n]$, is the number of times process P_i performs `update` followed by `scan`). Here, we use a vector to represent the number of rounds : this is because we do not want to treat only the layered immediate snapshot protocols, but more general atomic snapshot protocols. We claim now that the geometric semantics of the generic protocol, for n processes and (r) rounds, is represented by the pospace

$$\mathbb{X}_{(r)}^n = \prod_{i \in [n]} [0, r_i] \setminus \bigcup_{i, j \in [n], k \in [r_i], l \in [r_j]} U_i^k \cap S_j^l \quad (1)$$

- endowed with the product topology and product order induced by \mathbb{R}^n , where
- $n, r_i \in \mathbb{N}$ and u, s are any reals such that $0 < u < s < 1$: u (resp. v) is representing the local time at which an `update` (resp. `scan`) takes place in a round, and their precise values will not matter,
 - $U_i^k = \{x \in \prod_{i \in [n]} [0, r_i] \mid x_i = k + u\}$ stands for the region where the i -th process updates the global memory with its local memory for the k -th time,
 - $S_j^l = \{x \in \prod_{i \in [n]} [0, r_i] \mid x_j = l + s\}$ stands for the region where the j -th process scans the global memory into its local memory for the l -th time.

The meaning of (1) is that a state $(x_0, \dots, x_{n-1}) \in \prod_{i \in [n]} [0, r_i]$, i.e. a state in which each process P_i is at local time x_i , is allowed except when it is in $U_i^k \cap S_j^l$ (for $i, j \in [n]$ and $k \in [r_i], l \in [r_j]$): these forbidden states are precisely the states for which there is a `scan` and `update` conflict. Namely, states in $U_i^k \cap S_j^l$ are states for which process P_i updates (for the k -th time) while process P_j scans (for the l -th time), which is forbidden in the semantics. Indeed, the memory has to serialize the accesses since shared locations are concurrently read and written, and either the `scan` operation will come before the `update` one, or the contrary, but the two operations cannot occur at the same time. This is reflected in the geometric semantics by a hole in the state space,



as pictured on (2) for two processes with one round each, and in (4) for two processes with several rounds each. Notice that the holes are depicted as squares instead of points to improve the visibility on the diagram. In higher-dimensions, the holes exhibit a complicated combinatorics.

For instance, for three processes, and one round each, as in (3) shows forbidden regions that intersect one another. What happens in dimension 3 is that for all 3 pairs of processes (P,Q) , we have to produce a forbidden region which has a projection, on the two axes corresponding to P and Q , similar to the one on (2). Hence for all three pairs of processes, we have two cylinders with square section punching entirely the set of global states of the system. Each of these 6 cylinders correspond to a pair (P,Q) of processes, and a hole created either by a scan of P and an update of Q , or a scan of Q and an update of P . Consider the cylinder created by the conflict between the scan of P with the update of Q : it intersects exactly two cylinders (parallel to the other axes), the one created by the scan of the third processor R and the update of Q , and the one created by the update of R and the scan of P , see (3).

2.3 Equivalence of the standard and geometric semantics

In the geometric semantics of Section 2.2, we can define notions analogous to equivalence of traces as for the standard interleaving semantics of Section 2.1 (Proposition 1). A *dipath* (or *directed path*) in a pospace (\mathbb{X}, \leq) is a continuous map $\alpha : [0, 1] \rightarrow \mathbb{X}$ which is continuous and non decreasing when $[0, 1]$ is endowed with the order and topology induced by the real line. A dipath is the continuous counterpart (as we will make clear later) of a trace in the interleaving semantics, or an execution. A dipath $\alpha : [0, 1] \rightarrow \mathbb{X}$ is called *inextendible*, if there is no dipath $\beta : [0, 1] \rightarrow \mathbb{X}$ such that $\alpha([0, 1]) \subsetneq \beta([0, 1])$. This is the analogous, in our geometric setting, to maximal execution traces. The *concatenation* of two dipaths $\alpha, \alpha' : [0, 1] \rightarrow \mathbb{X}$ with compatible ends, i.e. $\alpha(1) = \alpha'(0)$ is the dipath $\alpha \cdot \alpha'$ such that $\alpha \cdot \alpha'(x)$ is $\alpha(x)$ (resp. $\alpha'(2x - 1)$) when $x \leq 0.5$ (resp. $x \geq 0.5$).

The continuous setting allows us to use the classical concepts of (di)homotopy, which is the natural notion of equivalence between paths, and to use some tools from algebraic topology to derive properties of protocols (and more generally programs [14]).

A *dihomotopy*

is a continuous map

$$H : [0, 1] \times [0, 1] \rightarrow \mathbb{X}$$

such that for all

$t \in [0, 1]$, the map

$H(-, t)$ is a dipath.

Two dipaths α, β such

that $\alpha(0) = \beta(0)$ and

$\alpha(1) = \beta(1)$ are *di-*

homotopic, if there is

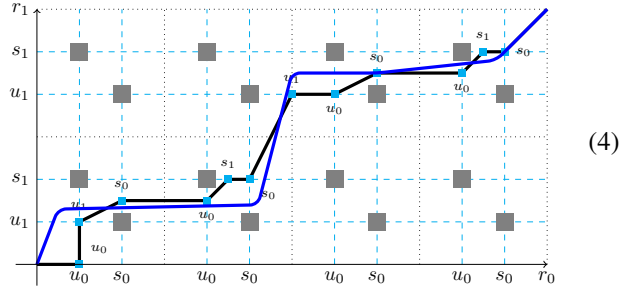
a dihomotopy $H :$

$$[0, 1] \times [0, 1] \rightarrow \mathbb{X}$$

with

$$H(-, 0) = \alpha \text{ and } H(-, 1) = \beta.$$

We denote by $[\alpha]$ the set of inextendible dipaths dihomotopic to α and $\text{dPath}(\mathbb{X})$ the set of dipaths up to dihomotopy. For instance, two dipaths that are dihomotopic in the geometric semantics $X_{(4,2)}^2$ can be pictured as in Figure (4).



From equivalence classes of interleaving traces to dipaths modulo dihomotopy. To any interleaving trace T with n processes and (r) rounds, we associate a dipath α_T in

$\mathbb{X}_{(r)}^n$. This dipath accurately reflects the whole computation of T , e.g. if T' extends T , then $\alpha_{T'}$ also extends α_T . For example, the black path of (4) is the dipath associated to the trace $u_0 u_1 s_0 u_0 s_1 s_0 u_1 u_0 s_0 u_0 s_1 s_0$: the points along it correspond to actions and the path consists of a linear interpolation between those. The dipath α_T is built by induction on the length of trace T : when T is of length 0, α_T is the constant dipath staying at the origin; when T is the concatenation of a trace T_1 with an action A , we concatenate the dipath α_{T_1} and a dipath β which is defined according to the previous actions in T_1 :

Lemma 3. *There exists a (not necessarily inextendible) dipath α_T in $\mathbb{X}_{(r)}^n$ such that $\alpha_T(0)_i = 0$, for every $i \in [n]$, and satisfying the following. For any $i \in [n]$, if the last action of process i in T is its k -th update, then $\alpha_T(1)_i \in \{k + u, k + \frac{u+s}{2}\}$. If it is its k -th scan, then $\alpha_T(1)_i \in \{k + s, k + 1\}$. If the last action in T is the k -th update of process i , then $\alpha_T(1)_i = k + u$. If it is the k -th scan of process i , then $\alpha_T(1)_i = k + s$.*

To a maximal interleaving trace T , we associate an inextendible dipath α'_T by further extending α_T to reach $(r_i)_{i \in [n]}$, the end of all inextendible dipaths in $\mathbb{X}_{(r)}^n$. Now,

Proposition 4. *Two equivalent interleaving traces induce dihomotopic dipaths.*

Equivalence between equivalence classes of interleaving traces and (colored) interval orders. In order to prove that dipaths modulo dihomotopy are in bijection with interleaving traces modulo equivalence, we introduce a combinatorial tool encoding the history of events observable on both an equivalence class of interleaving traces, and a dihomotopy class of dipaths in our continuous models.

Definition 5. *Let $(I_x)_{x \in X}$ be a family of intervals on the real line (\mathbb{R}, \leq) . This family induces a poset (X, \preceq) , where \prec is defined as $x \prec y$ if and only if for every $s \in I_x$ and $t \in I_y$ we have $s < t$. Such a poset is called an interval order [12]. We denote as $x \parallel y$ the independence relation.*

An $[n]$ -colored interval order is given by an interval order (X, \preceq) and a labeling function $\ell : X \rightarrow [n]$ such that two elements with the same label are comparable. Then for any $i \in [n]$, the restriction of the interval order to intervals labeled by i is a total order. We denote as $\mathbf{cIO}(X)$ the set of colored interval orders on a set X .

Proposition 6. *There is a bijection between $[n]$ -colored interval orders and traces up to equivalence.*

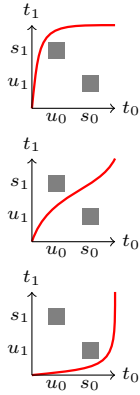
From Propositions 4 and 6, we can associate to any interval order a class of dipaths modulo dihomotopy. Let $i : \mathbf{cIO}(X_n) \rightarrow \mathbf{dPath}(\mathbb{X}_{(r)}^n)$ be mapping an interval order to a dipath up to dihomotopy.

From dipaths modulo dihomotopy to equivalence classes of interleaving traces. As already mentioned, dipaths geometrically represent execution traces, keeping in mind that dipaths which can be deformed through a continuous family of executions are operationally equivalent. This argument can be made concrete for the asynchronous model we are working on, by giving the explicit relation between dipaths and colored interval orders (Definition 5), because of Proposition 6.

To any inextendible dipath $\alpha : [0, 1] \rightarrow \mathbb{X}_{(r)}^n$, we associate an interval order \preceq_α on the set $X_{(r)}^n = \{(i, k) \mid i \in [n], k \in [r_i]\}$ through the interval collection for $i \in [n]$, $I_{(i,k)} = [u_i^k, s_i^k]$ colored by i where u_i^k or s_i^k respectively correspond to the event “ α enters an update or scan hyperplane”:

$$u_i^k = \inf \{t \in [0, 1] \mid \alpha(t)_i \in U_i^k\}, \quad s_i^k = \inf \{t \in [0, 1] \mid \alpha(t)_i \in S_i^k\}. \quad (5)$$

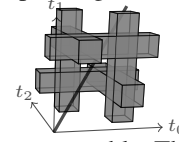
For any $i \in [n]$, the restriction of this order to the intervals labeled by i is a total order. Indeed, dipaths α are non decreasing, $u < s$ and $\alpha(u_i^k)_i = k + u$, $\alpha(s_i^k)_i = k + s$, hence for all $k \in [r_i]$, $u_i^k < s_i^k$ and if $k \neq 0$, $s_i^{k-1} < u_i^k$.



Let us give simple examples of this in dimension 2 and 3. In dimension 2, and for one round, consider the three inextendible dipaths in $\mathbb{X}_{(1,1)}^2$ pictured on the left (we are not writing the round number as upper index since we are considering here only one round). Those are representatives of the three dihomotopy classes of dipaths in this pospace. The dipath α_0 , on the above figure, corresponds to an execution in which process 1 does its update and scan before process 0 even starts updating. Hence, the interval of local times at which process 1 updates and scans is less than the interval of local times at which process 0 updates and scans: this is reflected by the corresponding interval order $[u_1, s_1] \prec_{\alpha_0} [u_0, s_0]$. The one on the figure below, α_Z is symmetric: the corresponding interval order is $[u_0, s_0] \prec_{\alpha_Z} [u_1, s_1]$. The dipath on the middle corresponds to an execution in which the two processes are running synchronously,

updating at the same time, and scanning at the same time: the corresponding interval order is $[u_0, s_0] \parallel [u_1, s_1]$.

In dimension 3, there are more dipaths that one can draw. Consider, for instance, the synchronous execution of the three processes (i.e. the pospace $\mathbb{X}_{(1,1,1)}^3$), shown on the right. It corresponds to the interval order where the intervals $[u_0, s_0]$, $[u_1, s_1]$ and $[u_2, s_2]$ are not comparable. The path figured corresponds to a synchronous execution.



We then have the following simple facts first :

Lemma 7. *Two inextendible dipaths α and β , which intersect the update and scan hyperplanes in the same order, are dihomotopic.*

We write $\alpha \rightsquigarrow \beta$ when the two dipaths are dihomotopic.

Proposition 8. *A dipath α is dihomotopic to the dipath associated to the interval order induced by α , that is, $i \circ r(\alpha) \rightsquigarrow \alpha$.*

This implies the following, among the main results of this article :

Proposition 9. *Two dihomotopic inextendible dipaths on $\mathbb{X}_{(r)}^n$ induce the same interval order.*

Theorem 10. *There is a bijective correspondence between traces up to equivalence and dipaths up to dihomotopy over $\mathbb{X}_{(r)}^n$, that is, $r : \mathbf{dPath}(\mathbb{X}_{(r)}^n) \rightarrow \mathbf{cIO}(X_n)$ which maps a dipath up to dihomotopy to an interval order, satisfies $r \circ i = \text{id}_{\mathbf{cIO}(X_n)}$.*

3 Protocol complexes, derived from the concurrent semantics

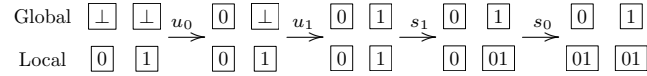
We will see that two executions modulo dihomotopy correspond to higher-dimensional simplices in protocol complexes (Proposition 13). In the case of update/scan protocols, these executions modulo dihomotopy are characterized by the nice combinatorial notion of interval order, which makes the construction of the protocol complex (Definition 17) from the geometric semantics immediate.

3.1 Protocol complex

The protocol complex has been designed [24] to represent the possible reachable states, at some given round, of the generic protocol in normal form, i.e. it is going to encode all possible histories of communication between processes, and as we will prove later on, all interleaving traces up to equivalence (or equivalently the dipaths up to dihomotopy), by maximal simplices :

Definition 11. *The protocol complex for atomic snapshot protocols is the abstract simplicial complex constructed from the generic protocol in normal form, and whose vertices are pairs (i, l_i) where $i \in [n]$ represents the name of a process and l_i its local memory, and whose maximal simplices are $\{(0, l_0), \dots, (n, l_n)\}$ where l_i is the local view by process i at the end of the execution represented by this simplex.*

Example 12. The local views in each vertex are determined by the operational semantics of Section 2.1, as in the following example:



leading to the local view $l = \langle\langle 0, \langle 0, 1 \rangle \rangle, \langle 1, \langle 0, 1 \rangle \rangle\rangle$. Similarly, the trace $u_0 s_0 u_1 s_1$ leads to the local view $l = \langle\langle 0, \langle 0, \perp \rangle \rangle, \langle 1, \langle 0, 1 \rangle \rangle\rangle$, and there is a third potential outcome of the computation, symmetric to this last case, in which process 1 updates and scans before process 0 does. Putting this together, according to Definition 11, we get the protocol complex for one round and two processes [24]:

$$0, (0\perp) \text{ --- } 1, (01) \text{ --- } 0, (01) \text{ --- } 1, (\perp 1)$$

The encoding of the local states, i.e. vertices in the graph above, is as follows. The identifier of the process whose local view is the number before the comma, e.g. the state $0, (0\perp)$ above is the local view of processor 0. The group of numbers or \perp within parentheses, e.g. $(0\perp)$ in the state above, is a condensed notation for the local state where $l_0 = \langle 0, \langle 0, \perp \rangle \rangle$, see Section 2.1. Similarly, state $1, (01)$ denotes the local view of processor 1, with local state such that $l_1 = \langle 1, \langle 0, 1 \rangle \rangle$.

3.2 Construction of the protocol complex from the directed geometric semantics

We can now link protocol complexes with interval orders, i.e. traces up to equivalence or dipaths up to dihomotopy: a colored interval order represents indeed an execution

and we can deduce the local view of the i -th process by restricting the interval order to the last scan of i . We encode local views restricting to the full information generic protocol in normal form with initial local state $l_i = i$ for $i \in [n]$ (this only changes the naming of local states, and not the structure of the protocol complex).

Proposition 13. *Let $(X_{(r)}^n, \preceq)$ be an $[n]$ -colored interval order. Then the local memory of the i -th process at round k of its corresponding execution³ is given by its restriction \mathcal{V}_i^k to the k -th scan S_i^k of the i -th process, i.e.*

$$\mathcal{V}_i^k = \{(j, l) \mid (i, k) \parallel (j, l) \text{ or } (j, l) \prec (i, k)\}$$

meaning that it is the value of the local state l_i under the semantics of Section 2.1 for the interleaving path corresponding to the interval order \mathcal{V}_i^k under the equivalence of Proposition 6.

Example 14. Consider again the one round, two processes case. We have represented below the protocol complex already depicted in Example 12, and decorated its maximal simplices, i.e. edges, with the corresponding dipaths modulo dihomotopy above, and the corresponding interval order, below:

$$0, (0\perp) \frac{\boxed{\begin{smallmatrix} \blacksquare \\ \blacksquare \end{smallmatrix}}}{0 \prec 1} 1, (01) \frac{\boxed{\begin{smallmatrix} \blacksquare \\ \blacksquare \end{smallmatrix}}}{0 \succ 1} 0, (01) \frac{\boxed{\begin{smallmatrix} \blacksquare \\ \blacksquare \end{smallmatrix}}}{0 \succ 1} 1, (\perp 1)$$

The local view of process 0 which is $0, (0\perp)$ comes from the restriction of the interval order $0 \prec 1$, subscript of the leftmost edge in the graph above, to 0: an interleaving trace corresponding to this interval order, under Proposition 6 is $u_0 s_0$ leading to local state $(0\perp)$ on process 0. Similarly, $1, (01)$ corresponds to the local state $l_1 = (01)$ for process 1, both for the restriction $0 \prec 1$ of $0 \prec 1$ to \mathcal{V}_1^1 (corresponding to a trace $u_0 s_0 u_1 s_1$, as in the trace $\boxed{\begin{smallmatrix} \blacksquare \\ \blacksquare \end{smallmatrix}}$ superscript of the edge on the left of the graph above) and for the restriction $0 \succ 1$ of $0 \succ 1$ to \mathcal{V}_1^1 (corresponding to a trace $u_0 u_1 s_0 s_1$ for instance, as in the trace $\boxed{\begin{smallmatrix} \blacksquare \\ \blacksquare \end{smallmatrix}}$ superscript of the middle edge of the graph above).

We are now in a position to give a combinatorial description of the protocol complex of Definition 11, using interval orders. We call the resulting equivalent complex, the *interval order complex*:

Definition 15. *The interval order complex is the simplicial complex whose*

- *vertices are $((i, k), V_i^k)$ where i stands for the i -th process, k for the round number and V_i^k for an interval order such that for all $(j, l) \in V_i^k$, either $(i, k) \parallel (j, l)$ or $(j, l) \prec (i, k)$,*
- *maximal simplices are $\{((0, r_0), V_0^{r_0}), \dots, ((n, r_n), V_n^{r_n})\}$ such that there is an interval order $(X_{(r)}^n, \prec)$ whose restriction to (i, r_i) is $V_i^{r_i}$.*

In that case we say that it is the interval order complex on (r) rounds and for $n + 1$ processes.

³ In the full-information generic protocol in normal form, i.e. its view, see Proposition 6 and the following example.

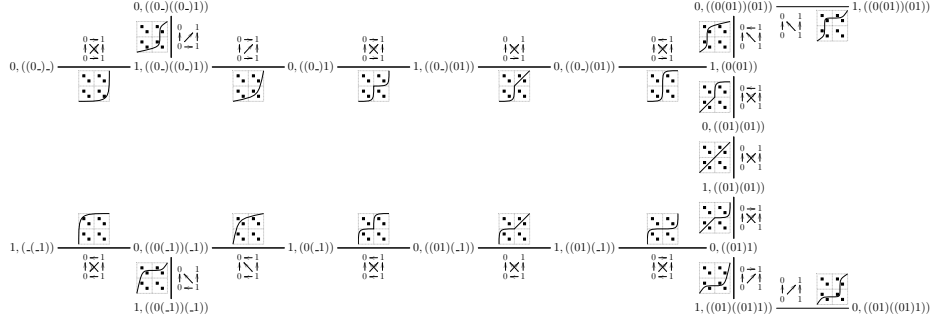


Fig. 1. Interval order complex, together with corresponding traces, of 2 processes, 2 rounds.

Example 16. An example of interval order complex with the traces corresponding to the execution for 2 processes, 2 rounds is depicted at Figure 1. Note that this is not the classical iterated subdivision in three parts at each round, i.e. a 9 edges complex, that is depicted for atomic snapshot protocols [22]. This is because we are considering more executions than the classical *layered immediate snapshot protocols* [22]: we allow round 2 of process 0 to begin while process 1 is still in round 1 for instance. Consider the interval order $\begin{array}{c} 0 \rightarrow 1 \\ \uparrow \times \uparrow \\ 0 \rightarrow 1 \end{array}$ labeling the upper left edge of the protocol complex in Figure 1, where an arrow $x \rightarrow y$ means $x \prec y$. As shown in the same figure, it corresponds to the execution $\begin{array}{|c|c|c|} \hline \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot \\ \hline \end{array}$ precisely where process 0 is executing its 2 rounds before process 1 even starts its first round. The local view of process 0 at (its) round 2 corresponds to the interval order $\begin{array}{c} 0 \\ \uparrow \\ 0 \end{array}$, restriction of $\begin{array}{c} 0 \rightarrow 1 \\ \uparrow \times \uparrow \\ 0 \rightarrow 1 \end{array}$ to $\mathcal{V}_0^{(2,0)}$. An interleaving trace corresponding to this is e.g. $u_0 s_0 u_0 s_0$, which, by the semantics of Section 2.1, leads to the local state of process 0: $\langle 0, \langle 0, \langle 0, \perp \rangle \rangle \perp$ written in condensed form as the upper left local state $0, ((0_)_)$ in Figure 1.

In Figure 2, we show the interval order complex for 3 processes and 1 round. Note again that we do not have exactly the same picture as in [22]: to the 13 triangles of [22], we have to add the 6 extra blue triangles that make the complex not faithfully representable as a planar shape and which correspond to non immediate snapshot executions. For instance, the upper left blue triangle is labeled with the interval order where 0 is not comparable to both 1 and 2, and 2 is less than 1. An interleaving trace (up to equivalence) corresponding to this interval order is given on the same figure: $u_0 u_2 s_2 u_1 s_1 s_0$.

3.3 Particular case of 1-round immediate snapshot protocols

We have not quite finished with describing the connections between directed algebraic topology and the protocol complex approach : the combinatorial description of the protocol complex in the case of layered immediate snapshot protocols seems, at first glance,

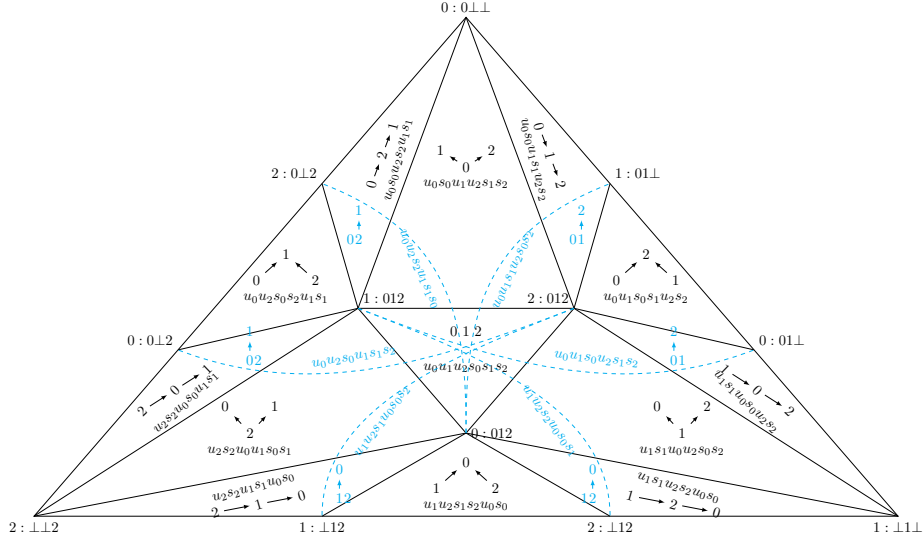


Fig. 2. Interval order complex with traces of 3 processes, 1 round.

of a different nature than the one using interval order complexes of Definition 15. We recall that an (layered, for multi-round protocols) *immediate snapshot* protocol [22] is a protocol where the snapshot of a given process comes “right after” its update, meaning that the allowed traces (within one round), up to equivalence, should be, of the form $u_{i_1} \dots u_{i_k} s_{i_1} \dots s_{i_k}$. Of course, there is some difference in that interval order complexes account for non necessarily layered nor “immediate” protocols. It is the aim of this section to make the connection between the subcomplex of interval order complexes describing layered immediate snapshot protocols, and the equivalent two definitions of chromatic barycentric subdivision [25,19] that describe combinatorially the protocol complex in that case.

The standard chromatic subdivision $\chi(\Delta^{[n]})$ of the standard $[n]$ -colored simplicial complex $\Delta^{[n]}$ is defined as follows (see [19], where an equivalence with the Definition in [25] is also shown):

Definition 17. *The standard chromatic subdivision $\chi(\Delta^{[n]})$ of $\Delta^{[n]}$ is the $[n]$ -colored simplicial complex whose vertices are pairs (V, i) with $V \subseteq [n]$ and $i \in V$ and simplices are sets of the form $\sigma = \{(V_0, i_0), \dots, (V_d, i_d)\}$ with $d \geq -1$ ($\sigma = \emptyset$ when $d = -1$) which are*

1. *well-colored: for every $k, l \in [d]$, $i_k = i_l$ implies $k = l$,*
2. *ordered: for every $k, l \in [d]$, $V_k \subseteq V_l$ or $V_l \subseteq V_k$,*
3. *transitive: for every $k, l \in [d]$, $i_l \in V_k$ implies $V_l \subseteq V_k$.*

This complex is colored via the second projection: $\ell(V, i) = i$.

Remark 18. The transitivity (property 3) of Definition 17 is equivalent to looking only at immediate snapshot executions. Observe the left upper blue triangle of Figure 1,

which is composed of vertices $(0 : 012)$, $(1 : 012)$ and $(2 : 0\perp 2)$ (respectively meaning $(\{0, 1, 2\}, 0)$, $(\{0, 1, 2\}, 1)$ and $(\{0, 2\}, 2)$ in the notations of Definition 17). It does not correspond to a layered execution: it corresponds to the equivalence class of traces $u_0u_2s_2u_1s_1s_0$. Transitivity does not hold either: $0 \in \{0, 2\}$ but $\{0, 1, 2\} \not\subseteq \{0, 2\}$. This leads us to the last main result of our article.

Theorem 19. *Layered immediate snapshot executions correspond to the interval orders such that: $J \prec K$ and I is not comparable with J implies $I \prec K$. The subcomplex of the interval order complex on one round, $(X_{(1,\dots,1)}^n, \preceq)$, that contains only immediate snapshot executions is isomorphic to the chromatic barycentric subdivision of Definition 17.*

4 Conclusion and future work

We have revealed strong connections between directed algebraic topology, with its applications to semantics and validation of concurrent systems, and the protocol complex approach to fault-tolerant distributed systems. This has been exemplified on the simple layered immediate snapshot model, but also on the more complicated (non layered, non immediate) iterated snapshot model. This, combined with the results of [26,19], entirely classifies geometrically the computability of wait-free layered immediate snapshot protocols, directly from the semantics of the update and scan primitives. We classified combinatorially, en route, the potential schedules of executions (equivalently, the potential local views of processes) as an interesting and well-known combinatorial structure: interval orders. This is a first step towards a more ambitious program. Fault-tolerant distributed models, whose protocol complex are more complex to guess combinatorially, may be handled by going through the very same steps we went through, starting with the geometric semantics of the communication primitives, and classifying dipaths modulo dihomotopy. We shall apply this to atomic read/write protocols with extra synchronization primitives such as test&set, compare&swap and others. In the long run, we would like to derive impossibility results directly by observing some obstructions in the semantics, in the form of suitable directed algebraic topological invariants.

Acknowledgments The first two authors were partially supported by the academic chair “Complex Systems Engineering” of École polytechnique-ENSTA-Télécom-Thalès-Dassault Aviation-DCNS-DGA-FX-Fondation ParisTech-FDO ENSTA. We also gratefully acknowledge Maurice Herlihy, Sergio Rajsbaum and Dmitry Kozlov for numerous discussions, and the referees for helping us improve this paper.

References

1. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4), September 1993.
2. J. H. Anderson. Composite registers. In *Conference on Principles of Distributed Computing*. ACM, New York, 1993.
3. O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *PoDC*. ACM, 1988.

4. R. Bonichon, G. Canet, L. Correnson, É. Goubault, E. Haucourt, M. Hirschowitz, S. Labbé, and S. Mimram. Rigorous evidence of freedom from concurrency faults in industrial control software. In *SAFECOMP*, 2011.
5. E. Borowsky and E. Gafni. Generalized FLP impossibility result for t -resilient asynchronous computations. In *STOC*, 1993.
6. J. Dubut, E. Goubault, and J. Goubault-Larrecq. Natural homology. In *Automata, Languages, and Programming - 42nd International Colloquium*, pages 171–183, 2015.
7. L. Fajstrup, É. Goubault, E. Haucourt, S. Mimram, and M. Raussen. Trace spaces: An efficient new technique for state-space reduction. In *ESOP*, 2012.
8. L. Fajstrup, É. Goubault, E. Haucourt, S. Mimram, and M. Raussen. *Directed Algebraic Topology and Concurrency*. Springer Verlag, 2015, to be published.
9. L. Fajstrup, É. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *CONCUR*, number 1466 in LNCS. Springer-Verlag, 1998.
10. L. Fajstrup, M. Raussen, and É. Goubault. Algebraic topology and concurrency. *TCS*, 357(1), 2006.
11. M. J Fischer, N. A Lynch, and M. S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.
12. P. C Fishburn. Intransitive indifference with unequal indifference intervals. *Journal of Mathematical Psychology*, 7(1):144–149, 1970.
13. G. Gierz. *A Compendium of continuous lattices*. Springer, 1980.
14. É. Goubault. Some geometric perspectives in concurrency theory. *Homology, Homotopy and Appl.*, 2003.
15. É. Goubault and E. Haucourt. A practical application of geometric semantics to static analysis of concurrent programs. In *CONCUR 2005*. Springer, 2005.
16. É. Goubault, T. Heindel, and S. Mimram. A geometric view of partial order reduction. *MFPS, Electr. Notes Theor. Comput. Sci.*, 298, 2013.
17. É. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR*, 1992.
18. É. Goubault. *The Geometry of Concurrency*. Ph.D. dissertation, ENS, 1995.
19. É. Goubault, S. Mimram, and C. Tasson. Iterated chromatic subdivisions are collapsible. *Applied Categorical Structures*, 2014.
20. M. Grandis. *Directed Algebraic Topology : Models of Non-Reversible Worlds*, volume 13 of *New Mathematical Monographs*. Cambridge University Press, 2009.
21. J. Gunawardena. Homotopy and concurrency. *Bulletin of the EATCS*, 54:184–193, 1994.
22. M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Elsevier, 2014.
23. M. Herlihy and N. Shavit. The asynchronous computability theorem for t -resilient tasks. In *STOC*. ACM, 1993.
24. M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM (JACM)*, 46(6):858–923, 1999.
25. D. Kozlov. Chromatic subdivision of a simplicial complex. *HHA*, 14(2), 2012.
26. D. Kozlov. Topology of the view complex. *arXiv preprint arXiv:1311.7283*, 2013.
27. M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4, 1987.
28. N. A Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.
29. L. Nachbin. *Topology and order*. Van Nostrand mathematical studies. Van Nostrand, 1965.
30. V. Pratt. Modeling concurrency with geometry. In *POPL*. ACM Press, 1991.
31. M. E. Saks and F. Zaharoglou. Wait-free k -set agreement is impossible: the topology of public knowledge. In *STOC*, 1993.
32. R. van Glabbeek. Bisimulation semantics for higher dimensional automata. Technical report, Stanford, 1991.