Distributed Decision Problems: Concurrent Specifications Beyond Binary Relations

Sergio Rajsbaum
Institute of Mathematics, National Autonomous University of Mexico, Mexico City, Mexico

Abstract

Much discussion exists about what is computation, but less about is a computational problem. Turing's definition of computation was based on computing functions. When we move from sequential computing to interactive computing, discussions concentrate on computations that do not terminate, overlooking notions of distributed problems. Many models where concurrency happens have been proposed, ranging from those equivalent to a Turing machine, to those where much heated discussion has taken place, claiming that interactive models are fundamentally different from Turing machines.

It is argued here that there is no need to go all the way to non-terminating interaction, to appreciate how different distributed computation is from sequential computation. The discussion concentrates on the various ways that exist of representing a *distributed decision problem.* Each process of a distributed system starts with an initial private input value, and after communicating with other processes in the system, produces a local output value. An input/output relation is needed, to specify which output values are legal for a particular assignment of input values to the processes.

An overview is provided of some results that show how rich the topic of distributed decision problems can be, when asynchronous processes can fail, but mostly independent of particular models of distributed computing and their many intricate details (types of failures and of communication). We are in a world very different from that of the functions of sequential computation; moving away from the world of graphs beyond binary relations, to the world of simplicial complexes.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed decision tasks, simplicial complex, linearizability, interval-linearizability, Arrow's impossibility, Speedup theorems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.3

Category Invited Talk

Funding Supported by UNAM-PAPIIT IN106520.

1 What is Computation and what is a Computational Problem?

The tools of a barber include scissors, razor, shave brush, comb, clipper, neck duster; the process that repeatedly uses these tools is barbering. It is awkward to talk about barbering before saying what the problem being solved is: shaving, hair-cutting, and hair-dressing. Yet, it seems we are sometimes more obsessed with understanding what is computation, than with understanding what is a computational problem.

The first ACM Ubiquity symposium (2011) thoroughly discussed the question: What is computation? The most fundamental question of our field, says Peter Denning in the Editor's Introduction [18]. But except for mentioning Turing and how he invented his machine to classify *functions* according to computability, not much is said about computational problems.

For sequential computing not much is discussed about computational problems, beyond functions, and for distributed computing even less. The participants of the workshop were asked to consider how three new developments might have affected the traditional answers to the question. One of the three developments is interactive computation, motivated by situations such as operating systems and networks that are based on computations that do

33rd International Conference on Concurrency Theory (CONCUR 2022).
Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 3; pp. 3:1–3:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

not terminate and regularly interact with their environments. All through the discussions in the workshop, it seems that the interest on interactive computation comes from their non-terminating nature, e.g. [24].

Some argue about the enduring legacy of the Turing Machine like Lance Fortnow [21], while others strongly against it, like Peter Wegner [54]. But in the conclusions of the workshop, Denning [17] mentions that there is an emerging consensus that interactive models are fundamentally different from Turing machines.

Aho [1] easily describes the computational problem: A function f from strings to strings is computable if there is some Turing machine M that given any input string w always halts in the accepting state with just $f(w)$ on its tape. But describes in detail what a Turing machine is:

> *The reason we went through this explanation is to point out how much detail is involved in precisely defining the term computation for the Turing machine, one of the simplest models of computation. It is not surprising, then, as we move to more complex models, the amount of effort needed to precisely formulate computation in terms of those models grows substantially.*

Aho [1] continues: Many real-world computational systems compute more than just a single function – the world has moved to interactive computing. But there is no discussion of what is it that they compute.

Indeed, as the authors of the workshop discuss, there are many models of distributed computing, consisting of autonomous computing processes that communicate with one another. To model multicore shared memory systems, wide area message passing networks, biological systems such as cells and organisms, even the human brain. There are theoretical models such as message-passing Actor model, Petri nets, process calculi, I/O automata, etc. Many shared-memory and message passing models are discussed in the distributed computing literature, e.g. [5, 33, 48, 49].

2 Distributed Decision Problems

To discuss distributed computing problems, very few details about the computational model need to be considered; the same notions of distributed computing problem are relevant to many of the models mentioned above.

2.1 Distributed computing problems

There are many problems to discuss about distributed computing. Distributed systems can exhibit behaviors such as deadlock, livelock, race conditions. And there are many aspects to study about routing, robot coordination, agents moving along a network, distributed graph algorithms, and the like that cannot be studied using Turing machines. Concerns such as reliability, performance, scalability and adaptivity, mobility, psychical locality, are inherently different from sequential computing.

All through the symposium, it is emphasized the importance of models where interaction takes place, assuming as evident that the interest is in non-terminating computations. I would like to slow down here, to show the richness exhibited already in *terminating* distributed computation. Furthermore, that there is no need to get into the intricacies of a distributed computing model, to discuss distributed problems. The goal is to show that indeed very novel issues arise that do not exists in Turing machines, already when we consider input/output problems.

In this paper the goal is to focus on possibly the purest form of distributed computing problem, a direct analogue of the notion of a function for a Turing machine. The input x to the function is now distributed, each process knows only part of x. Also the output $f(x)$ is distributed: after communicating with each other, the processes collectively compute $f(x)$, each one computes one part of it. As we shall see, instead of functions it is of interest to consider relations $T(x)$, called *tasks*, possibly allowing for more than one output for each input x.

Assume the simplest case of a fixed, finite set of n individual processes composing the distributed system. To focus only on the problem of computing a task in a distributed way, disregard any routing and network communication problems, and assume that the processes can directly communicate with each other. Similarly, to focus only on the distributed aspects of the problem, disregard any individual sequential computing limitation. It turns out that some tasks have no solution, even if each process is an infinite state automata, while when there is a solution, each process is a (usually) simple Turing machine.

For the purposes of discussing distributed problems, there is no need to discuss many of the specifics about the computational model – ways in which processes communicate with each other, their relative speeds and failures. Roughly, the only thing needed, is that a process may have to produce an output value without knowing the input values of some of the other processes.

2.2 Distributed decision tasks

Early on in the development of distributed computing theory, Moran and Wolfstahl [43] defined the notion of *distributed decision task*, to encompass the various problems that were being studied at that time, such as consensus, approximate agreement and renaming. It was already known that consensus is impossible to solve in a message passing system even if only one process can fail by crashing [20] (even if each process is an infinite state machine). Moran and Wolfstahl extended the impossibility to general decision tasks, and then Biran, Moran and Zaks [6] extended it to a full characterization.

Consider n-dimensional vectors with entries over some set of possible values V: the i-th entry of a vector is associated to the i-th process. A distributed decision task $\mathcal{T} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ consists of a set of *input vectors*, $\mathcal{I}$ a set of *output vectors*, $\mathcal{O}$ and a relation Δ, specifying, for each input vector $I \in \mathcal{I}$, a set of legal output vectors $\Delta(I) \subseteq \mathcal{O}$. The i-th entry of an input vector is the input value of the i-th process. The i-th entry of an output vector is the output value of the i-th process. It is assumed that each process, has two special variables, a read-only one for the input value and a write-once variable for the output value.

A decision task is *solvable* by a distributed algorithm in some model of computation, if the following holds. The system can start in any of the input vectors $I \in \mathcal{I}$ allowed by the task. Now, consider any execution starting with input vector I, where all processes produce an output value, defining a vector O consisting of all the n output values. Then, it must be the case that $O \in \Delta(I)$.

Notice that task solvability is defined only by a safety requirement. There is also a liveness requirement defined by the specifics of the model of computation. In the sequel of papers by Biran, Moran, Zaks and Wolfstahl [6, 7, 8, 43], the focus was on 1-resilient asynchronous processes (running at arbitrary speeds, independent from each other) communicating by message passing. In this case, the liveness requirement is that, in an execution where at most one process crashes, all processes that do not crash have to produce a decision value. A similar situation but in shared memory was considered by Moran and Taubenfeld [53], including the case where $t < n$ processes may crash, where the liveness is adjusted accordingly.

The following examples are well-known by now.

1. *Consensus.* For a set of values V, the inputs are all n-vectors over V. There is one output vector for each $v \in V$, consisting of all output values equal to v, denoted O_v, and $\mathcal{O} = \cup_{v \in V}\{O_v\}$. For any input vector with at least two different input values, $\Delta(I) = \mathcal{O}$, for an input vector I_v with a single input value v, $\Delta(I_v) = \{O_v\}$.
2. *Approximate agreement.* It is defined in [6] for any given $\epsilon > 0$, and V the set of rational numbers. Any n-vector over V is a possible input vector, and the output vectors contain rational numbers so that for any two entries d_i, d_j, $|d_i - d_j| \leq \epsilon$. Then, $\Delta(I)$ contains all output vectors with entries d_i such that $m \leq d_i \leq M$, where m is the smallest value of I and M is the largest.

There are many variants of consensus and approximate agreement, including multidimensional ones e.g. [41].

2.3 Participating processes

The discovery of the intimate connection between distributed computing and topology, overviewed in [30], was facilitated by the realization that the 1-resilient case is not the most fundamental situation, and surprisingly not the easiest to analyze – it is the *wait-free* case. Wait-freedom is a progress condition which guarantees that each process can make progress in a finite number of steps regardless of the behavior of other processes. So long as processes are scheduled, wait-freedom guarantees progress for all processes. Thus, a distributed algorithm that is wait-free never includes instructions by which a process waits for an event of another process (if that process crashes, the event might never happen).

For this paper, the important property is that any set of processes may have to produce output values, without knowing the input values of the remaining processes. Therefore, the vectors of a decision task need to incorporate a notion of *participating* processes. Not all entries in a given input (output) vector need contain an input (output) value; some may contain the special value $\perp$, indicating that some processes do not participate in the execution (crashes before taking any steps). Thus, the set of input vectors is required to be prefix closed. Meaning that if I is an input vector, then the task has to consider also any input vector I' contained in I, in the sense that any subset of the entries of I is replaced by $\perp$. Furthermore, for each such input vector I', where the input values of some subset of the processes P' is defined as $\perp$, the input/output relation Δ has to specify what are the legal output vectors. Namely, $\Delta(I')$ is a set of vectors, all with $\perp$ in the entries for processes P'.

As already discussed by Herlihy and Shavit [32] and Hoest and Shavit [35], the intuitive notion of "order of actions in time" is captured through the use of participating processes. The example given is how it can be used to distinguish between tasks such as Unique-Id and Fetch-And-Increment, which have the same sets of input and output vectors, have the same Δ when all processes participate, but have quite different task specification maps when subsets of participating processes are taken into account. The Figure 1 is from [35], for a set of $n + 1$ processes.

The Unique-Id task is defined as follows: each participating process $i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses an output $y_i \in \{0, \ldots, n\}$ such that for any pair of processes $i \neq j$, $y_i \neq y_j$.

In the Fetch-And-Increment task, each participating process $i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses a unique output $y_i \in \{0, \ldots, n\}$ such that (1) for some participating process i, $y_i = 0$, and (2) for $1 \leq k \leq n$, if $y_i = k$, then for some $j \neq i$, $y_j = k - 1$.

$(0,\bot,\bot)$	$(0,\bot,\bot),(1,\bot,\bot),(2,\bot,\bot)$
$(\bot,0,\bot)$	$(\bot,0,\bot),(\bot,1,\bot),(\bot,2,\bot)$
$(\bot,\bot,0)$	$(\bot,\bot,0),(\bot,\bot,1),(\bot,\bot,2)$
$(0,0,\bot)$	$(0,1,\bot),(1,0,\bot),(0,2,\bot),(0,2,\bot),(2,1,\bot),(1,2,\bot)$
$(0,\bot,0)$	$(0,\bot,1),(1,\bot,0),(0,\bot,2),(0,\bot,2),(2,\bot,1),(1,\bot,2)$
$(\bot,0,0)$	$(\bot,0,1),(\bot,1,0),(\bot,0,2),(\bot,0,2),(\bot,2,1),(\bot,1,2)$
$(0,0,0)$	$(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$

(a) Unique-Id task.

$(0,\bot,\bot)$	$(0,\bot,\bot)$
$(\bot,0,\bot)$	$(\bot,0,\bot)$
$(\bot,\bot,0)$	$(\bot,\bot,0)$
$(0,0,\bot)$	$(0,1,\bot),(1,0,\bot)$
$(0,\bot,0)$	$(0,\bot,1),(1,\bot,0)$
$(\bot,0,0)$	$(\bot,0,1),(\bot,1,0)$
$(0,0,0)$	$(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$

(b) Fetch-And-Increment task.

Figure 1 Two tasks with the same set of vectors when all participate. First column is the input vector, and for each row I, in the second column $\Delta(I)$.

2.4 Beyond binary relations

These two examples already hint at why by moving from vectors to partial vectors, the notion of decision task is interestingly enriched. This is clearly exposed using the appropriate mathematical structure for partial vectors closed under containment: simplicial complexes. Here follows and overview of how to use them to represent tasks, additional details are in e.g. [30].

The following notions are illustrated in Figure 2, where the Fetch-And-Increment task is represented using simplicial complexes, for three processes denoted $0,1,2$. Intuitively, triangles represent vectors with 0 entries equal to $\bot$, edges represent vectors with 1 entry equal to $\bot$, and vertices correspond to vectors with 0 entires equal to $\bot$. An input vertex (i,x) means that process i has input value x, and for the vertex $(0,0)$, $\Delta(i,x)) = \sigma_1$. For the input edge $\{(0,0),(1,0)\}$, $\Delta(\{(0,0),(1,0)\}) = \{\sigma_2,\sigma_3\}$, while for the input triangle $\{(0,0),(1,0),(2,0)\}$, $\Delta(\{(0,0),(1,0),(2,0)\})$ consists of all 6 triangles of $\mathcal{O}$.

A *simplicial complex* is a generalization of a graph, where sets of vertices of cardinality more than two can also be grouped into a *simplex* (the generalization of an edge). Formally, it is a collection $\mathcal{K}$ of non-empty sets, closed under containment, i.e., if $\sigma \in \mathcal{K}$ then, for every non-empty set $\sigma' \subseteq \sigma$, $\sigma' \in \mathcal{K}$. Every set in $\mathcal{K}$ is called a *simplex*. A subset of a simplex is called a *face*, and a *facet* of $\mathcal{K}$ is a face that is maximal for inclusion in $\mathcal{K}$. The *dimension* of a simplex σ is $|\sigma| - 1$, where $|\sigma|$ denotes the cardinality of σ. The dimension of a complex is the maximal dimension of its facets. A complex in which all facets are of the same dimension is called *pure*. The *vertices* of $\mathcal{K}$ are all simplices with a single element (i.e., of dimension 0). The set of vertices of a complex $\mathcal{K}$ are denoted by $V(\mathcal{K})$.

All complexes in this paper are *chromatic*, i.e., every vertex is a pair $v = (i,x)$ where $i \in [n] = \{1,\dots,n\}$ for some $n \geq 1$ is the color of v denoting a process, and x is some value (an input value or an output value). Moreover, in a chromatic complex, a color i must appear at most once in every simplex. Let $\sigma = \{(i,x_i) : i \in I\}$ be a simplex. We denote by $\text{ID}(\sigma)$ the set of colors in σ, i.e., $\text{ID}(\sigma) = I$. Indeed, in the following, the color of a vertex is actually the *identity* of a process.

A task can be defined using vectors as above, or using simplicial complexes as follows, exposing the role of combinatorial topology notation, and why going beyond binary relations is intrinsic to distributed computing problems.

A *task* for n processes is a triple $\Pi = (\mathcal{I},\mathcal{O},\Delta)$ where $\mathcal{I}$ and $\mathcal{O}$ are $(n-1)$-dimensional complexes, respectively called *input* and *output* complexes, and $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is an input-output specification. Every simplex $\sigma = \{(i,x_i) : i \in I\}$ of $\mathcal{I}$, where $I = \text{ID}(\sigma)$ is a non-empty subset of $[n]$, defines a legal input state corresponding to the scenario in which, for every $i \in I$, process i starts with input value x_i. Similarly, every simplex $\tau = \{(i,y_i) : i \in I\}$ of $\mathcal{O}$ defines a legal output state corresponding to the scenario in which, for every $i \in I$, process i outputs the value y_i. The map Δ is an input-output relation specifying, for every input state

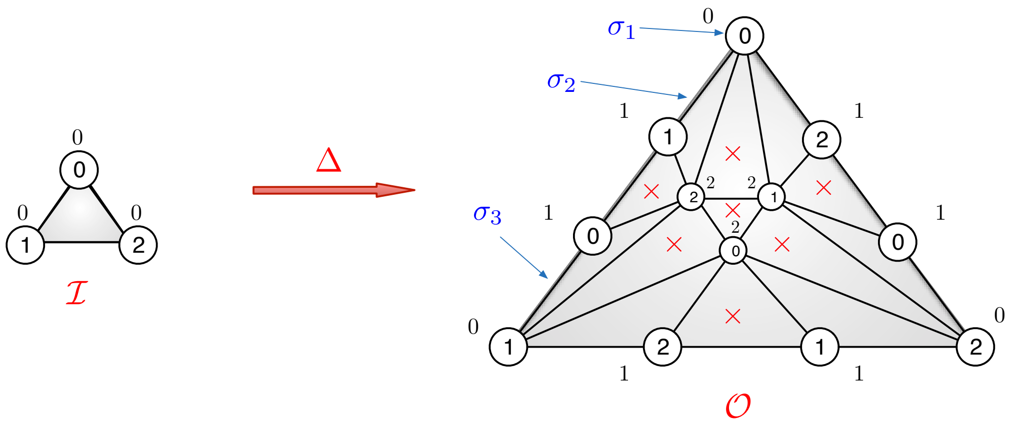

Figure 2 The Fetch-And-Increment task. Inside a vertex is its id, outside is its input or output value. The input complex $\mathcal{I}$ consists of a single triangle, and its faces. The input complex $\mathcal{O}$ consists of 6 triangles, and its faces. The triangles marked with an $\times$ are deleted.

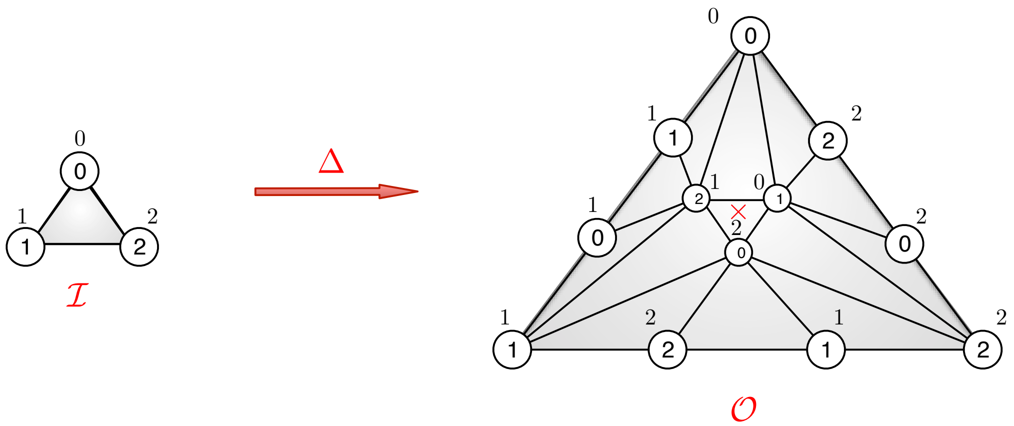

Figure 3 The input complex of the set agreement task for 3 processes, and part of the output complex. The triangle marked with $\times$ is deleted. The corners of the input triangle are mapped by Δ to the corners of $\mathcal{O}$. The boundary of the input triangle is mapped to the boundary of $\mathcal{O}$. The input triangle is mapped to all the depicted 12 triangles of $\mathcal{O}$.

$\sigma \in \mathcal{I}$, the set of output states $\tau \in \mathcal{O}$ with $\text{ID}(\tau) = \text{ID}(\sigma)$ that are legal with respect to σ. That is, assuming that only the processes in $\text{ID}(\sigma)$ participate to the computation (the set of participating processes is not known a priori to the processes in σ), these processes are allowed to output any simplex $\tau \in \Delta(\sigma)$. It is often assumed that Δ is a carrier map (that is, for every $\sigma, \sigma' \in \mathcal{I}$, if $\sigma' \subseteq \sigma$ then $\Delta(\sigma') \subseteq \Delta(\sigma)$ as subcomplexes).

An important example is the *set agreement* task, with a single input facet (and all its faces), where process i starts with input value i. The n processes need to agree on at most $n-1$ input values of participating processes. For three processes, at most two different values can be decided, as illustrated in Figure 3, where part of the output complex is depicted. This task is important, because it is unsolvable wait-free [9, 32, 52], and the reason for the impossibility is a topological one: intuitively, the task has a hole while no wait-free distributed algorithm has one.

3 Selected Topics

Here is a selection of the various aspects about tasks that have been studied. Consensus is the most fundamental task, in a sense the most difficult one together with variants such as *interactive consistency* where all of the processes have to agree on the same vector such that the ith entry of the vector contains the value proposed by the i-process; any task is solvable, if processes can agree on their inputs. Much can be said about consensus in long-lived situations, and consensus is known to be enormously important in real systems since early on [37], as well as in theory e.g. [29], for reasons including the consensus hierarchy [39] and as a universal object [50], but here the focus is on decision problems.

3.1 Colorless tasks, local tasks, continuous tasks: decidability and reductions

The class of *colorless tasks* was identified in [10]. Such a task can be defined in terms of sets of input and output values, without referring to which process is assigned which input value or produces which output value, and without referring to the number of processes in the system. Many widely-studied tasks are colorless, including consensus, set-agreement, and approximate agreement. Some important tasks like renaming [13] and others [12] are not colorless, and are more difficult to study, but easier than set agreement [11]. A notion of *continuous task* has been prosed aiming at obtaining wait-free solvability characterization [25] in a more intuitive way than the original one [32].

The *rendezvous task* [38] is a colorless task that models scenarios where autonomous agents move around in a specific space to meet one another. A chromatic version where a process must end in a vertex of its own color in a chromatic subdivision of an input simplex, is the *chromatic simplex agreement task,* important for the wait-free task solvability theorem [32], and the *affine tasks*, on subcomplexes of the chromatic subdivision by Kuznetsov and Rieutord [36]. The *loop agreement task* is an example of rendezvous task, which is defined in terms of an edge loop in a 2-complex. Herlihy and Rajsbaum [31] showed that a loop agreement task is wait-free solvable if and only if the loop is contractible in the 2-complex, as a result, the wait-free solvability of loop agreement tasks is undecidable. Rendezvous on the vertices of a graph was introduced in [15], and variants were studied in [3] including applications to robot coordination problems [2].

A task G *implements* task F if one can construct a protocol for F by calling any protocol for G, possibly followed by access to a shared read/write memory. This notion of implementation induces a partial order on tasks and hence it induces a classification of a set of tasks, into disjoint classes such that tasks in the same class implement each other. In this sense, all tasks in a class are computationally equivalent. A classification of loop agreement tasks was presented in [31], and extended in [55] to rendezvous tasks.

A task T is wait-free checkable if and only if it satisfies a certain *locality* condition. Notions of locality considered by Fraigniaud, Travers and Rajsbaum [23] are mostly independent of the computing model. Wait-free solvability of local tasks remains undecidable. A strong notion of locality is defined by *covering tasks* whose output complex is a covering of the input complex. This topological property yields obstacles for wait-free solvability different in nature from the classical agreement impossibility results, and, apart from the identity task, locality-preserving tasks are not wait-free solvable. A classification of locality-preserving tasks in term of their computational power is presented. Also closely related to covering tasks and with a similar impossibility argument [26], is the *equality negation task.* For two processes, each of which has an input from a set of three distinct values, each process must decide a binary output value so that the decisions of the processes are the same if and only if the initial values of the processes are different. This task was defined by Lo and Hadzilacos [39], as the central idea to prove that the consensus hierarchy is not robust.

Fraigniaud, Paz and Rajsbaum [22] study consensus and approximate agreement, through an approach for proving lower bounds and impossibility results, called the *asynchronous speedup theorem.* For a given task T and a given computational model M, the *closure* of T with respect to M is a task that is supposed to be a slightly easier version of T. The asynchronous speedup theorem states that if a task T is solvable in $t \geq 1$ rounds in M, then its closure w.r.t. M is solvable in $t-1$ rounds in M. As an application they study the power of test&set and binary consensus, for wait-free solving approximate agreement faster.

3.2 Domain restrictions and social choice

A research line started by Mostefaoui, Rajsbaum and Raynal [44] considers restricting the input domain of a task, to obtain an easier task. A restriction of the input complex is called a *condition.* For example, although consensus is unsolvable even if only one process can crash, if we assume that more than a majority of processes propose the same value then consensus becomes solvable ($n \geq 4$). The paper identified the conditions for which consensus is solvable in an asynchronous distributed system with t crash failures. In a sequel paper [45] they study conditions for consensus in a synchronous system where processes can fail by crashing. A hierarchy of conditions parametrized by d is presented, that allows solving synchronous consensus with less and less rounds, as we go from $d = t$ to $d = 0$.

There are remarkable analogies between social choice theory and distributed computing, despite the fact that social choice theory is typically not concerned with concurrency (for decentralised studies see [16, 40]). The modern field of social choice theory took off with Kenneth Arrow's remarkable 1950 result [4] for the basic problem of democracy: it is impossible to aggregate individual preferences into a single social preference, under some reasonable-looking axioms. In Arrow's setting, each process proposes a total order on the possible candidates, and the outcome of the election, computed by a centralized aggregation function f, is also a total order, that should reflect the social preference. One requirement is *unanimity*, if everyone prefers candidate x over y, so should the social preference. With only this requirement, the aggregation function can simply decide on the preferences of one individual, say the 1st one, which would become a dictator. Arrow's impossibility says that f must be dictatorial, if one requires, additionally to the unanimity requirement, an *independence of irrelevant alternatives* (IIS) requirement, stating that f depends only on pairwise preferences.

Much research has been devoted to identify domain restriction to circumvent Arrow's impossibility theorem. Rajsbaum and Raventós [47] identify the exact domain restrictions for the case of two voters and three alternatives, and present a new proof of Arrow's impossibility based on a task formalization using simplicial complexes, showing that any unanimous IIS aggregation function must be dictatorial, on any of the corresponding restricted domains. The proof uses techniques analogous to those used in distributed computing [13, 26].

3.3 Tasks and objects

Tasks are not the only possible input/output distributed specifications. Objects are defined in terms of sequential specifications, and can specify ongoing, never-ending behavior, such as for concurrent data structures [42]. For this paper consider their one-shot version, and one method that can be invoked only once by each process, with an input parameter. The object returns an output value to the invoking process. Thus, one-shot objects are similar to tasks, in that they specify input/output problems. Objects come with an accompanying notion of when a concurrent execution satisfies the object's sequential specification, linearizability.

In fact we encounter in the literature three ways of talking about distributed decision problems. As a set of informal requirements, as a sequential object plus a consistency condition (linearizability), and as a task. For example, we have seen that consensus can be defined as a task. But often it is defined by two safety requirements. Validity: a decided value is the input of some participating process; Agreement: any two decided values are equal. The third way is to think of consensus as an object, defined by a sequential automata, whose states represent which values have been proposed to the object, and which values can be returned to a process.

The relation between tasks and objects has been studied by Castañeda, Rajsbaum and Raynal [14], motivated by Neiger [46], who proposed a generalization of linearizability to be able to specify tasks, such as set agreement, which have no natural specification as sequential objects. Set-sequential objects can define executions in which a set of processes access an object concurrently. The notion of an *interval-sequential* object [14], together with a corresponding consistency condition, is able to express any concurrency pattern of overlapping invocations of operations, that might occur in an execution [27]. While some important tasks have no specification either as a sequential object nor as a set-sequential object, all tasks can be naturally expressed as interval-sequential objects. Remarkably, there are objects that cannot be expressed as tasks. An extension of the task framework is described, called *refined tasks,* that has more expressive power, and is able to specify any one-shot interval-sequential object.

An interesting notion appears with objects, *composabilty*, which has not been studied as much for tasks. Linearizability is very popular to design components of large systems because one can consider linearizable object implementations in isolation and compose them for free, without sacrificing linearizability of the whole system [34]. It was shown that by going from linearizability to interval-linearizability, one does not sacrifice the benefits of composability [14].

3.4 Tasks and knowledge

Rosenbloom [51] argues that computation is information transformation. In this sense, one may view a distributed problem as setting the goals, from an initial state of information, to a final one. More precisely, a task is reformulated by Goubault, Rajsbaum and Ledent [28] as a knowledge transformation goal. The input complex defines what processes know about each other inputs, formalized as a *simplicial model*, the dual of the classic one-dimensional Kripke model, that exposes relations beyond binary. A task can be re-interpreted as a goal in terms of knowledge gain, using an output simplicial model, which is the product update of the initial simplicial model and an action model. This formally specifies the knowledge gain required by the task.

The importance of common knowledge for reaching agreement is well understood [19]. Consensus and common knowledge is discussed in [28], as well as approximate agreement, in terms of knowledge gain. After all, the difficulty of distributed decisions comes from the absence of common knowledge about the inputs and consensus gives us just that. Indeed, in the epistemic setting, consensus is the requirement of achieving common knowledge on an input value. This is impossible in asynchronous systems. In contrast, approximate agreement is solvable, because it is a finite version of common knowledge, requiring only that everybody knows that everybody knows, and so on, a certain number of times.

References

1 Alfred V. Aho. Computation and Computational Thinking. *The Computer Journal*, 55(7):832–835, July 2012. doi:10.1093/comjnl/bxs074.

2 Manuel Alcantara, Armando Castañeda, David Flores-Peñaloza, and Sergio Rajsbaum. The topology of look-compute-move robot wait-free algorithms with hard termination. *Distributed Comput.*, 32(3):235–255, 2019. doi:10.1007/s00446-018-0345-3.

3 Dan Alistarh, Faith Ellen, and Joel Rybicki. Wait-free approximate agreement on graphs. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2021. doi:10.1007/978-3-030-79527-6_6.

4 Kenneth J. Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58(4):328–346, 1950. doi:10.1086/256963.

5 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, Hoboken, NJ, USA, 2004.

6 Ofer Biran, Shlomo Moran, and Shmuel Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *J. Algorithms*, 11(3):420–440, 1990. Preliminary version in PODC 1988. doi:10.1016/0196-6774(90)90020-F.

7 Ofer Biran, Shlomo Moran, and Shmuel Zaks. Deciding 1-sovability of distributed task is np-hard. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 16rd International Workshop, WG '90, Berlin, Germany, June 20-22, 1990, Proceedings*, volume 484 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 1990. doi:10.1007/3-540-53832-1_44.

8 Ofer Biran, Shlomo Moran, and Shmuel Zaks. Tight bounds on the round complexity of distributed 1-solvable tasks. In Jan van Leeuwen and Nicola Santoro, editors, *Distributed Algorithms, 4th International Workshop, WDAG '90, Bari, Italy, September 24-26, 1990, Proceedings*, volume 486 of *Lecture Notes in Computer Science*, pages 373–389. Springer, 1990. doi:10.1007/3-540-54099-7_25.

9 Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 91–100. ACM, 1993. doi:10.1145/167088.167119.

10 Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Comput.*, 14(3):127–146, 2001. doi:10.1007/PL00008933.

11 Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Renaming is weaker than set agreement but for perfect renaming: A map of sub-consensus tasks. In David Fernández-Baca, editor, *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, volume 7256 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2012. doi:10.1007/978-3-642-29344-3_13.

12 Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Generalized symmetry breaking tasks and nondeterminism in concurrent objects. *SIAM J. Comput.*, 45(2):379–414, 2016. doi:10.1137/130936828.

13 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Comput. Sci. Rev.*, 5(3):229–251, 2011. doi:10.1016/j.cosrev.2011.04.001.

14 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. Unifying concurrent objects and distributed tasks: Interval-linearizability. *J. ACM*, 65(6):45:1–45:42, 2018. doi:10.1145/3266457.

15 Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Two convergence problems for robots on graphs. In *2016 Seventh Latin-American Symposium on Dependable Computing, LADC 2016, Cali, Colombia, October 19-21, 2016*, pages 81–90. IEEE Computer Society, 2016. doi:10.1109/LADC.2016.21.

16 Himanshu Chauhan and Vijay K. Garg. Democratic elections in faulty distributed systems. In Davide Frey, Michel Raynal, Saswati Sarkar, Rudrapatna K. Shyamasundar, and Prasun Sinha, editors, *Distributed Computing and Networking, 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings*, volume 7730 of *Lecture Notes in Computer Science*, pages 176–191. Springer, 2013. doi:10.1007/978-3-642-35668-1_13.

17 Peter J. Denning. Reflections on a Symposium on Computation. *The Computer Journal*, 55(7):799–802, July 2012. doi:10.1093/comjnl/bxs064.

18 Peter J. Denning and Peter Wegner. Introduction to What is Computation. *The Computer Journal*, 55(7):803–804, July 2012. doi:10.1093/comjnl/bxs065.

19 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001.

20 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

21 Lance Fortnow. The enduring legacy of the turing machine. *Comput. J.*, 55(7):830–831, July 2012. doi:10.1093/comjnl/bxs073.

22 Pierre Fraigniaud, Ami Paz, and Sergio Rajsbaum. A speedup theorem for asynchronous computation with applications to consensus and approximate agreement. *To appear in ACM PODC 2022*, abs/2206.05356, 2022. To appear in ACM PODC 2022. doi:10.48550/arXiv.2206.05356.

23 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. *Distributed Comput.*, 26(4):223–242, 2013. doi:10.1007/s00446-013-0188-x.

24 Dennis J. Frailey. Computation is Process. *The Computer Journal*, 55(7):817–819, July 2012. doi:10.1093/comjnl/bxs069.

25 Hugo Rincon Galeana, Sergio Rajsbaum, and Ulrich Schmid. Continuous tasks and the asynchronous computability theorem. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 73:1–73:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.73.

26 Éric Goubault, Marijana Lazic, Jérémy Ledent, and Sergio Rajsbaum. Wait-free solvability of equality negation tasks. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 21:1–21:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.DISC.2019.21.

27 Éric Goubault, Jérémy Ledent, and Samuel Mimram. Concurrent specifications beyond linearizability. In Jiannong Cao, Faith Ellen, Luis Rodrigues, and Bernardo Ferreira, editors, *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*, volume 125 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.OPODIS.2018.28.

28 Éric Goubault, Jérémy Ledent, and Sergio Rajsbaum. A simplicial complex model for dynamic epistemic logic to study distributed task computability. *Inf. Comput.*, 278:104597, 2021. doi:10.1016/j.ic.2020.104597.

29 Rachid Guerraoui, Michel Hurfin, Achour Mostéfaoui, Rui Carlos Oliveira, Michel Raynal, and André Schiper. Consensus in asynchronous distributed systems: A concise guided tour. In Sacha Krakowiak and Santosh K. Shrivastava, editors, *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, volume 1752 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 1999. doi:10.1007/3-540-46475-1_2.

30 Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013. URL: https://store.elsevier.com/product.jsp?isbn=9780124045781.

31 Maurice Herlihy and Sergio Rajsbaum. A classification of wait-free loop agreement tasks. *Theor. Comput. Sci.*, 291(1):55–77, 2003. doi:10.1016/S0304-3975(01)00396-6.

32 Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999. doi:10.1145/331524.331529.

33 Maurice Herlihy and Nir Shavit. *The art of multiprocessor programming*. Morgan Kaufmann, 2008.

34 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

35 Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006. doi:10.1137/S0097539701397412.

36 Petr Kuznetsov and Thibault Rieutord. Affine tasks for k-test-and-set. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 151–166, Cham, 2020. Springer International Publishing.

37 Butler Lampson. How to build a highly available system using consensus. In Ozalp Babaoglu and Keith Marzullo, editors, *10th International Workshop on Distributed Algorithms (WDAG 1996)*, pages 1–17. Springer, October 1996. The proceedings are: Distributed Algorithms, Lecture Notes in Computer Science 1151, Springer, 1996. URL: https://www.microsoft.com/en-us/research/publication/how-to-build-a-highly-available-system-using-consensus/.

38 Xingwu Liu, Zhiwei Xu, and Jianzhong Pan. Classifying rendezvous tasks of arbitrary dimension. *Theor. Comput. Sci.*, 410(21-23):2162–2173, 2009. doi:10.1016/j.tcs.2009.01.033.

39 Wai-Kau Lo and Vassos Hadzilacos. All of us are smarter than any of us: Nondeterministic wait-free hierarchies are not robust. *SIAM J. Comput.*, 30(3):689–728, 2000. doi:10.1137/S0097539798335766.

40 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Byzantine preferential voting. In George Christodoulou and Tobias Harks, editors, *Web and Internet Economics*, pages 327–340, Cham, 2018. Springer International Publishing.

41 Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K. Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015. doi:10.1007/s00446-014-0240-5.

42 Mark Moir and Nir Shavit. Concurrent data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch47.

43 Shlomo Moran and Yaron Wolfstahl. Extended impossibility results for asynchronous complete networks. *Inf. Process. Lett.*, 26(3):145–151, 1987. doi:10.1016/0020-0190(87)90052-4.

44 Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *J. ACM*, 50(6):922–954, 2003. doi:10.1145/950620.950624.

45 Achour Mostefaoui, Sergio Rajsbaum, and Michel Raynal. Synchronous condition-based consensus. *Distributed Computing*, 18(5):325–343, 2006. doi:10.1007/s00446-005-0148-1.

46 Gil Neiger. Set-linearizability. In James H. Anderson, David Peleg, and Elizabeth Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, page 396. ACM, 1994. doi:10.1145/197917.198176.

47 Sergio Rajsbaum and Armajac Raventós-Pujol. A distributed combinatorial topology approach to arrow's impossibility theorem. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC '22, page to appear, New York, NY, USA, 2022. Association for Computing Machinery.

48 Michel Raynal. *Concurrent Programming - Algorithms, Principles, and Foundations*. Springer, 2013. doi:10.1007/978-3-642-32027-9.

49 Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018. doi:10.1007/978-3-319-94141-7.

50 Michel Raynal. The notion of universality in crash-prone asynchronous message-passing systems: A tutorial. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 334–33416, 2019. doi:10.1109/SRDS47363.2019.00046.

51 Paul S. Rosenbloom. Computing and Computation. *The Computer Journal*, 55(7):820–824, July 2012. doi:10.1093/comjnl/bxs070.

52 Michael E. Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000. doi:10.1137/S0097539796307698.

53 Gadi Taubenfeld and Shlomo Moran. Possibility and impossibility results in a shared memory environment. *Acta Informatica*, 33(1):1–20, 1996. Preliminary version in WDAG 1989. doi:10.1007/s002360050034.

54 Peter Wegner. The Evolution of Computation. *The Computer Journal*, 55(7):811–813, July 2012. doi:10.1093/comjnl/bxs067.

55 Yunguang Yue, Jie Wu, and Fengchun Lei. The evolution of non-degenerate and degenerate rendezvous tasks. *Topology and its Applications*, 264:187–200, 2019. doi:10.1016/j.topol.2019.06.015.