Towards Domain Theory for Distributed Tasks

Jérémy Ledent
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Roman Kniazev
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Abstract

The asynchronous computability theorem provides a criterion of solvability of distributed tasks in terms of existence of simplicial maps. In this paper, we devise an alternative version of this theorem based on directed-complete partial orders, structures routinely used in denotational semantics of programming languages. By doing so, we move away from combinatorial and point-set topology, exhibiting at the same time that they are both recoverable from the partial order structure.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Domain Theory, Distributed Computing, Distributed Tasks, Asynchronous Computability Theorem

Related Version This work was submitted as a Brief Announcement to the DISC 2024 conference.

1 Introduction

The combinatorial topology approach to distributed computing has been a powerful tool for studying solvability of fundamental distributed problems such a consensus, set agreement and renaming [5]. The starting point of this approach was the *Asynchronous Computability Theorem* (ACT) of Herlihy and Shavit [6], which establishes a correspondence between a computational question, solvability of a distributed task, and a geometric one, the existence of a map between simplicial complexes. Since then, several variants of the theorem have been formulated, to work in a variety of settings: e.g., t-resilient protocols and non-compact models [4, 1], fair adversaries [7], or allowing communication via arbitrary shared objects [8].

One approach that is particularly relevant to our paper is the one of [1]. They noticed that in the presence of infinite executions, the simplicial structure becomes disconnected and is not able to characterize task solvability anymore. So, they introduced a metric on the set of infinite histories of the model, thus recovering the underlying topology.This approach, where computation histories are considered explicitly, brings some operational flavor back to the simplicial framework.

In this paper, we follow this lead and formulate an asynchronous computability theorem that gets rid of the simplicial complex altogether. Instead of using a metric space, we equip the set of possible executions of a protocol with a directed-complete partial order structure. We show that the simplicial structure, as well as the point-set topology of [1], can be recovered from the directed-complete partial order. For the time being, we restrict ourselves to the usual setting of wait-free computation, but we hope that in the future this approach will enable us to accommodate more complex models.

2 Preliminaries

In this section, we recall the minimal mathematical background necessary to read this paper. A thorough account of domain theory and its links with topology can be found in [2].

A *partially ordered set*, or *poset*, consists of a set P equipped with a binary relation $\leq$ that is reflexive, transitive, and antisymmetric. Fix a poset $(P, \leq)$ and a subset $S \subseteq P$. An element $a \in P$ is called an *upper bound* of S if for all $x \in S$, $x \leq a$. An element $b \in P$ is called the *supremum* (a.k.a. least upper-bound) of S, if b is an upper bound, and moreover $b \leq a$ for every upper bound a of S. When a supremum exists, it is unique, and we write it $b = \sup S$.

Given a poset $(P, \leq)$, a subset $D \subseteq P$ is called *directed* if it is nonempty and every pair of elements has an upper bound : $D \neq \varnothing$ and for all $x, y \in D$, there is $z \in D$ such that $x \leq z$ and $y \leq z$. A *directed-complete partial order*, or *dcpo* for short, is a poset $(P, \leq)$ with the additional property that every directed subset D must have a supremum. A function $f : P \to Q$ between dcpos is called *(Scott-)continuous* if it preserves directed suprema, that is, if for every directed subset D of P the equality $\sup f(D) = f(\sup D)$ holds. Note that this definition implies, in particular, that f must be monotone.

▶ **Example 1** (Domain of histories)**.** Fix a set A whose elements are called *actions*. A *history* on A is a finite-or-infinite sequence of elements of A. We write $\mathcal{H} = A^* \cup A^\omega$ for the set of histories on A. The *prefix* ordering $\preceq$ on histories is defined by $h \preceq h'$ iff there exists h'' such that $h \cdot h'' = h'$, where $\cdot$ denotes concatenation. Then $(\mathcal{H}, \preceq)$ is a dcpo: a directed set on $\mathcal{H}$ is a (possibly infinite) chain of histories $h_0 \preceq h_1 \preceq h_2 \preceq \ldots$, where each history extends the previous one. The supremum of such a chain is either its maximal element (if the chain is finite), or the infinite history that has all the histories $(h_i)_{i \in \mathbb{N}}$ as prefixes, otherwise.

▶ **Example 2** (Finite poset)**.** A degenerate example is given by any poset $(P, \leq)$ where the set P is finite. There, any directed set has a maximal element, which is the supremum.

▶ **Example 3** (Flat domain)**.** Fix a set S. A *flat domain* $S_\perp$ consists of a set $S \cup \{\perp\}$, $\perp \notin S$, such that for any $s \in S$, $\perp \leq s$ and $s \leq s$, and all other elements incomparable.

For a poset P, its *opposite* is the poset P^{op} with the same elements, but with the order relation reversed, that is $x \leq^{\mathsf{op}} y$ in P^{op} if and only if $y \leq x$ in P. Additionally, we will denote the two-element poset $\{0 \leq 1\}$ by $\mathsf{2}$.

▶ **Definition 4** (Monotone relation)**.** *A* monotone relation *between two posets P and Q is a monotone function $R : P \times Q^{\mathsf{op}} \to \mathsf{2}$.*

Equivalently, a monotone relation can be seen as a subset R of $P \times Q$, such that (i) if $(x, y) \in R$ and $x \leq x'$, then $(x', y) \in R$; and (ii) if $(x, y) \in R$ and $y' \leq y$, then $(x, y') \in R$.

3 Domains and task solvability

In this section, we assume the reader is familiar with standard definitions of distributed protocols, distributed tasks and task solvability. See for example [5] for a detailed account in the setting of combinatorial topology.

Consider n processes, and let $\Pi = [n]$ be the set of process ids. For each $i \in \Pi$, suppose there is a set $\mathsf{P_i}$ of *local states* for process i. A *protocol* is given by a set P of global histories (as in Example 1), and for each process $i \in \Pi$, there is a function $\lambda_i : \mathsf{P} \to \mathsf{P_i}$ mapping each history $h \in \mathsf{P}$ to the local state of process i after executing h.

▶ Remark 5. This setting is closely related to the one of *interpreted systems* [3].

3.1 Task solvability via domains

Protocols as dcpos. First, note that the set of histories P can be equipped with the prefix order to yield a dcpo, as in Example 1. We may think of the order $\preceq$ on P as modelling a notion of causal dependency: as the computation progresses in time, we can only go up in the order relation on P.

We also assume that we have a dcpo structure on the sets of local states $\mathsf{P_i}$ for each process $i \in \Pi$, such that the projections $\lambda_i : \mathsf{P} \to \mathsf{P_i}$ are continuous (in particular, monotone). This is a fairly strong restriction, as processes are not allowed to go back to a previous local state. However, this is always possible as long as we consider a *full-information protocol*, where a process always recalls its local view of the history.

▶ **Example 6** (Lossy-link model)**.** Consider a synchronous message-passing model for two processes, where at each round at most one message can be lost. Thus, at each round, there are three possible types of executions: both processes hear from each other; only process 0 receives a message; only process 1 receives a message. The initial fragment of the dcpo P representing this model is depicted in Figure 1a: each node represents a global state (an element of P), and there is an arrow $x \to y$ between two elements if $x \leq y$.

▶ **Example 7** (Asynchronous read-write registers)**.** We now consider an asynchronous shared-memory model for two processes, and two shared single-writer, multi-reader registers. Each process starts by writing its value in its own register, then reads the value stored in the other process's register. Initially, registers contain a special "blank" value.

The corresponding dcpo is depicted in Figure 1b. Note that this example is not a tree: we have merged some histories that lead to the same global state (for example, when both processes write their value, the order does not matter).

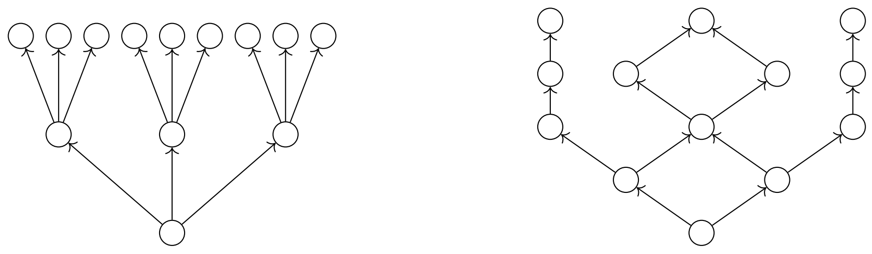
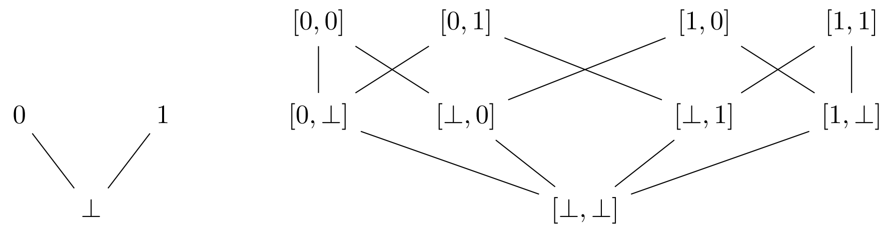

(a) First two rounds in the lossy-link model. **(b)** One round read-write register model.

Figure 1 Two different protocols viewed as dcpos.

Tasks as dcpos. Let Val be a set of possible decision values. We write $\mathsf{Val}_\perp$ for the corresponding flat domain (see Example 3), where the extra $\perp$ value will stand for "undecided". This value $\perp$ will be assigned to partial executions where processes do not have enough information to decide. Since we have n processes, we need to consider n-tuples of decision values as a dcpo: we write $(\mathsf{Val}_\perp)^n$ for the n-fold product of $\mathsf{Val}_\perp$ with itself. Concretely, its elements are tuples $(v_i)_{i\in\Pi}$ where each $v_i \in \mathsf{Val} \cup \{\perp\}$. These tuples are ordered componentwise: $(u_i)_{i\in\Pi} \leq (v_i)_{i\in\Pi}$ iff for every $i \in \Pi$, either $u_i = \perp$ or $u_i = v_i$.

▶ **Example 8** (Binary decision values)**.** Consider only two possible decision values, $\mathsf{Val} = \{0, 1\}$, and $n = 2$ processes. In the picture below, the dcpo $\mathsf{Val}_\perp$ is shown on the left, and the product $(\mathsf{Val}_\perp)^2$ is shown on the right.



In order to specify a task, we first need to select a subset of O of $(\mathsf{Val}_\perp)^n$, which will represent the allowed combinations of decision values. Note that O is still a dcpo, and it inherits the product structure from $(\mathsf{Val}_\perp)^n$: for every $i \in \Pi$, there is a projection $\pi_i : \mathsf{O} \to \mathsf{Val}_\perp$ indicating the decision of process i.

▶ **Example 9** (Binary consensus)**.** To model the binary consensus task, processes should agree on the same decision value, so only the tuples $[0,0]$ and $[1,1]$ are allowed. For two processes, the corresponding dcpo $\mathsf{O} \subseteq (\mathsf{Val}_\perp)^2$ is depicted below.

$$\begin{tikzcd}
 & {[0,0]} \arrow[dl, no head] \arrow[d, no head] & & {[1,1]} \arrow[d, no head] \arrow[dr, no head] & \\
{[0,\perp]} \arrow[drr, no head] & {[\perp,0]} \arrow[dr, no head] & & {[\perp,1]} \arrow[dl, no head] & {[1,\perp]} \arrow[dll, no head] \\
 & & {[\perp,\perp]} & &
\end{tikzcd}$$

In order to have a task specification, we still need to say how decision values relate to input values. Here, we take a slightly non-standard approach, in that we do not explicitly define the input values. Instead, we let $I \subseteq \mathsf{P}$ be the minimal elements of P, i.e., the initial (global) states of the system. Thus, implicitly, we are assuming that each process starts the computation with just its input value as a local state: given an initial state $s \in I \subseteq \mathsf{P}$, the input value of process i is $\lambda_i(s) \in P_i$.

▶ **Definition 10** (Task specification)**.** *A task specification is a monotone relation* $\Delta \subseteq \mathsf{P} \times \mathsf{O}$*.*

So the task specification relates global states of the system, with tuples of allowed decision values. In the above definition, monotonicity says two things:

- Given $(s, v) \in \Delta$, that is, an initial state $s \in \mathsf{P}$ is compatible with a tuple of decisions $v \in \mathsf{O}$, any history s' that is reachable from s must also be compatible with v.
- Additionally, once again assuming that $(s, v) \in \Delta$, any other tuple of decisions $v' \leq v$ which is less-defined that v, must also be compatible with s.

The second item might seem too permissive: at any point in the protocol, the processes are always allowed to "decide $\perp$", i.e., they may remain undecided forever. This is because the task specification only indicates which combinations of inputs and outputs are allowed. The halting criterion (wait-free, obstruction-free, etc.) will be defined separately.

Task solvability. In order to solve the task, each process $i \in \Pi$ has its own local decision function $\delta_i : \mathsf{P_i} \to \mathsf{Val}_\perp$ that sends each local state either to $\perp$ ("undecided"), or to a decision value $v_i \in \mathsf{Val}$. Every δ_i captures the dynamic aspect of execution by assigning $\perp$ to partial executions where the process cannot decide yet. These local decisions assemble into a global decision function $\delta : \mathsf{P} \to (\mathsf{Val}_\perp)^n$, defined by $\delta(s) = (\delta_i(\lambda_i(s)))_{i \in \Pi}$, as depicted in the diagram below.

$$\begin{tikzcd}
\mathsf{P} \arrow[r, "\lambda_\mathsf{i}"] \arrow[d, "\delta"'] & \mathsf{P_i} \arrow[d, "\delta_\mathsf{i}"] \\
(\mathsf{Val}_\perp)^n \arrow[r, "\pi_\mathsf{i}"'] & \mathsf{Val}_\perp
\end{tikzcd}$$

Then, the protocol solves the task Δ if (i) $\delta : \mathsf{P} \to \mathsf{O}$, i.e., at every state the tuple of decisions is among the allowed outputs; and (ii) $\delta \subseteq \Delta$, i.e., the decision values respect the task specification: for every $s \in \mathsf{P}$, $(s, \delta(s)) \in \Delta$.

If we do not assume that the decision functions δ_i are given in advance, the previous discussion turns into the following theorem:

▶ **Theorem 11** (Asynchronous Computability Theorem for Domains). *The protocol* P *solves the task* Δ *if there is a continuous function* $\delta : \mathsf{P} \to \mathsf{O}$ *such that* $\delta \subseteq \Delta$*, that is, if* $\delta(s) = v$*, then* $(s, v) \in \Delta$*.*

In the statement of the theorem, the word "continuous" does the heavy lifting. First, δ must be monotone, which means that as time progresses, we can only go up in the dcpo O. Hence, processes cannot undo their decisions. Moreover, continuity ensures that in every execution, a process must decide in finite time. Indeed, infinite executions in P are suprema of their finite prefixes. Continuity states that $\delta(\sup D) = \sup \delta(D)$: the decision of an infinite trace is the supremum of the decisions of its finite prefixes. But since the dcpo of outputs O is finite, the supremum is actually a maximum, so the decision must have been reached in one of the prefixes.

Note that we are still missing a halting condition: the function δ of Theorem 11 can send every state to $\bot$. We can define wait-free implementations as an extra condition on δ. Again, continuity plays a role, to ensure that decisions are reached in finite time.

▶ **Definition 12** (Wait-free implementation). *A function* $\delta : \mathsf{P} \to \mathsf{O}$ *is a* wait-free *implementation if every maximal element of* P *is sent to a maximal element of* O*. In particular, for the dcpo of histories, for every infinite history* h *and every process* $i \in \Pi$*,* $\pi_i(\delta(h)) \neq \bot$*.*

3.2 Links with other approaches

Recovering the simplicial structure. Given a dcpo P together with maps $\lambda_i : \mathsf{P} \to \mathsf{P_i}$ for each process $i \in \Pi$, we can recover the usual simplicial simplex structure as follows. The set of vertices of color i is defined to be $\mathsf{P_i}$. Furthermore, for every maximal element $p \in \mathsf{P}$, there is an n-simplex $\{\lambda_i(p) \mid i \in \Pi\}$.

Recovering point-set topology. Given a dcpo P, one can define the *Scott topology* on P as follows. Its open sets are the upper-closed sets $U \subseteq P$ such that moreover, whenever $\sup D \in U$ for some directed set D, there is $d \in D$ such that $d \in U$.

It is again instructive to consider the case of the domain of histories from Example 1. Since every open set is upper-closed, it contains an infinite history. But since infinite sequences are suprema of directed subsets, there must be a finite prefix that belongs to the open.

▶ **Proposition 13.** *The Scott topology on* P_i *restricted to infinite executions coincides with the topology from [1].*

4 Conclusion

We have defined the notions of distributed protocols, task specifications and task solvability in terms of Scott domains. This approach could lead to new tools and techniques to study distributed computability. We hope that our setting can be generalized in several ways. We have defined wait-free implementations, and it seems possible to similarly define obstruction-free, or lock-free implementations. For example, one can explicitly model crashes by adding to the local dcpos of processes a special element fail which is greater than any other element. Another point of improvement would be to lift some of the technical restrictions that we have imposed: we had to restrict to full-information protocols; and the dcpo structure prevents us from modeling non-compact models, as done in [1]. Finally, we hope that our approach might be able to go beyond distributed tasks and model implementability of long-lived objects.

References

1 Hagit Attiya, Armando Castañeda, and Thomas Nowak. Topological characterization of task solvability in general models of computation. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023*, volume 281 of *LIPIcs*, pages 5:1–5:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.DISC.2023.5`, `doi:10.4230/LIPICS.DISC.2023.5`.

2 Ulrich Berger, Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove, and Dana S. Scott. Continuous lattices and domains. *Stud Logica*, 86(1):137–138, 2007. URL: `https://doi.org/10.1007/s11225-007-9052-y`, `doi:10.1007/S11225-007-9052-Y`.

3 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995. URL: `https://doi.org/10.7551/mitpress/5803.001.0001`, `doi:10.7551/MITPRESS/5803.001.0001`.

4 Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 222–231. ACM, 2014. `doi:10.1145/2611462.2611477`.

5 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`.

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

7 Petr Kuznetsov, Thibault Rieutord, and Yuan He. An asynchronous computability theorem for fair adversaries. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 387–396. ACM, 2018. URL: `https://dl.acm.org/citation.cfm?id=3212765`.

8 Jérémy Ledent and Samuel Mimram. A sound foundation for the topological approach to task solvability. In *30th International Conference on Concurrency Theory, CONCUR 2019*, page 34:1–34:15, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.34`.