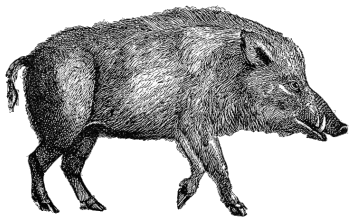
Li, Kwing Hei

Wait-Free Task Solvability of Asynchronous Distributed Models

Master of Philosophy in
Advanced Computer Science

King's College

June, 2023



Declaration

I, Li Kwing Hei of King's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed 

Date June 1, 2023

Acknowledgements

Marcelo Fiore is an excellent supervisor who goes above and beyond to support his students.

Maurice Herlihy and **Christine Tasson** answered many of my questions with much clarity and just as much patience.

The illustration on the cover is a hat tip to **Martin Kleppmann**'s book [12]. Martin's courses on distributed systems was what got me interested in this subject in the first place.

My master's degree is funded by the **Cambridge Trust & King's College TPP Alan Turing Scholarship**.

Abstract

As modern programs become more concurrent, we need to design good mathematical models to reason about distributed algorithms. This report concerns two **incompatible** models studied by two separate lines of research on asynchronous distributed systems. On the one hand, we have **HKR's model** [9], whose task solvability power is well-studied and has deep connections with combinatorial topology, as captured by the Asynchronous Computability Theorem [10]. On the other hand, we have **GMT's model** [8], which is proven to possess a range of elegant geometric semantics.

The contributions of this report are twofold.

Firstly, we develop a **novel unified mathematical framework** that enables us to reason about both models at the same time. This framework is powerful enough to model various aspects of the distributed system, e.g. protocol types, execution traces, and solvability properties.

Secondly, as a significant proof of concept, we utilize this framework to prove the Fundamental Theorem of Asynchronous Distributed Models, a **new result** that shows that a task description specification is solvable by HKR's model if and only if it is solvable by GMT's model. This fundamental theorem is the first result to unite the results of the two lines of research, allowing results on task solvability of one model to be shared with the other. For example, the Asynchronous Computability Theorem can be trivially extended to describe the task solvability power of GMT's model, in addition to HKR's model.

No AI-assisted tools employing generative Large Language Models are used in this project; they all lack the necessary creativity and imagination to make the proofs work.

Contents

Chapter 1

Introduction

In Computability Theory, we are often interested in determining whether a problem is decidable or not, i.e. if there exists an algorithm that always leads to a yes-or-no answer for solving the problem. For example, the halting problem [3] is widely known to be undecidable over Turing machines (or any other Turing-complete model of computation), and one can subsequently prove the undecidablity of other problems via a reduction argument.

However, modern software does not normally consist of single machines running individually, but a large ecosystem of machines running in tandem while communicating with each other via interfaces, e.g. reading and writing on shared memory, message passing through channels. From the small multi-core processor to the large wide-area network, one can find concurrency *everywhere*.

In light of this, one would naturally aim to generalize the original decidability problem in order to account for this concurrency aspect. Suppose we have a finite set of "machines" interacting with some sort of communication primitive. What kind of distributed tasks are solvable? Is there any mathematical property behind these distributed tasks that makes them solvable, or perhaps unsolvable?

1.1 Background

The solvability of distributed tasks has been explored by various researchers beginning as early as the 1980s. Perhaps the most famous result in this field is the FLP impossiblity result [4], which proves the long-standing conjecture that fault-tolerance consensus is unachievable in an asynchronous message-passing model.

In this report, we highlight two lines of research on asynchronous distributed systems.

The first is that of **Herlihy et al.** [9]. Herlihy and his collaborators found deep connections between the solvability of distributed tasks and structures in combinatorial topology. In particular, the Asynchronous Computability Theorem [10] precisely presents a sufficient and necessary condition to the topological structure of a distributed task, in order for it to be solvable by a very specific shared-memory distributed model, which we refer to as **HKR's model** in this report.

On the other hand, **Goubault et al.** [8] generalized various aspects of HKR's model, and established various geometric semantics for their model. We call their model **GMT's model**. However, the task solvability power of GMT's model has neither been explored nor studied. Instead, only three propositions on task solvability were presented in the paper [8], which were not proved. The authors only pointed out that the proofs could be found in HKR's work. This, however, is not satisfactory, especially because GMT's definition of task solvability, model, and trace execution differs from

HKR's, and it is not obvious why HKR's proofs would work for GMT's propositions. In fact, one of the propositions is not true if one uses HKR's definition of task solvability (see subsection 5.2.1).

To summarize, we now have two **incompatible** definitions of asynchronous distributed systems. The task solvability power of the former model is well understood, while the latter is more general and is proven to have elegant geometric semantics. There is, however, no literature as to why and how both models are equivalent, or even related to one another. This report strives to fill this gap by showing that the task solvability power of both models is **equivalent**.

1.2 Contributions

The aim of this document is to study the task solvability power of GMT's model and how it relates to HKR's model. We do this in two steps.

Firstly, building upon GMT's work, we develop a **unified mathematical framework** that enables one to reason about both GMT's and HKR's models, deriving various reduction proofs within this new framework. This framework is powerful enough to model various aspects of a distributed system, e.g. protocol types, execution traces, solvability properties.

Secondly, we demonstrate the flexibility and power of our framework by **proving the following theorem**, a *new* result that relates the task solvability power of GMT's and HKR's models:

Theorem (Fundamental Theorem of Asynchronous Distributed Models (Simplified))**.** Given task description Θ and solvability property P, Θ is P-solvable by GMT's model if and only if Θ is P-solvable by HKR's model.

Informally, a task description is the specification of the problem we want our distributed model to solve, i.e. a set of valid input-output pairs. A solvability property is a predicate which depends on the task description and various aspects of the model, defining what it means for a model to solve the distributed problem.

This theorem is a testimony to the success of our unified theory of asynchronous distributed systems. It succinctly describes how the task solvability of GMT's abstract distributed model is equivalent to HKR's specialized model from the Asynchronous Computability Theorem. Note that unlike all past results, the statement of our fundamental theorem, and subsequently the proofs, are specifically designed to be **task solvability-agnostic**. This means the theorem holds for any definition of solvability property, as long as it is a predicate of the correct type (see subsection 2.2.5). This important feature means that the result is general enough to hold for both GMT's and HKR's definition of task solvability, as well as any other instances of the solvability property.

Since this unified framework is novel, all proofs presented here are also novel. A few proofs are inspired by known algorithms (which the author acknowledges explicitly in later sections) but the treatment of lifting these algorithms to fit our framework remains novel.

In chapter 2, we describe the basic framework of our report, and use it to define our most general definition of an asynchronous distributed system, i.e. GMT's model. Chapter 3 examines how various aspects of our model can be specialized to account for more interesting models, and we use a combination of these variants to define HKR's model. After introducing both models, in chapter 4, we prove the above fundamental theorem, by showing how a protocol of one model can be transformed to another via various intermediate forms, and similarly, vice versa. Lastly, in chapter 5, we present various directions for enriching our unified theory of asynchronous distributed systems.

Let us begin.

Chapter 2

Preliminaries

In this chapter, we examine the basics of the theoretical framework we use for the rest of this report.

We start off by presenting some assumptions on the distributed system we want to model with our framework which should capture aspects of a general asynchronous distributed system. We then formally present various components of our model, while proving relevant properties along the way. We present the definition of our most general base model, i.e. GMT's model, and conclude by discussing how our base model is faithful to our assumptions we present at the start of this chapter.

2.1 Assumptions

We make the following base assumptions about our distributed system. These assumptions capture a general abstract notion of an asynchronous distributed system.

1. The system contains a **finite** number of processes.
2. The sets of valid input and output values are **finite**.
3. Processes communicate via a shared-memory array through two operations: an **atomic read** in which the process scans the entire array and updates their local state, and a **write** in which the process updates its own memory cell according to its local state.
4. The set of processes participating in the protocol is **arbitrary**. The set of allowed outputs of the overall system might also be different depending on the initial set of participating processes. However, this set is not revealed to the processes beforehand.
5. Participating processes might **crash** at any point of the execution, and never recover afterwards. Together with the previous assumption, one can say the number of active processes can only decrease during the execution trace.
6. The model is **asynchronous**, i.e. there are no synchronization primitives on the processes, allowing them to execute at different speeds.
7. Once a process changes its local state to a valid output value, it cannot perform any other actions, other than crashing. We call this a **commit**.
8. The program executed by each process is **wait-free** [11], i.e. each non-faulted process must be able to commit in a finite amount of steps regardless of the actions of other processes.

2.2 Model Definition

Given our assumptions, we now present the formal definition and semantics of our framework's model. Note that our definition is heavily inspired by that of GMT's work [8] as the author finds their notation to be more familiar to the working computer scientist.

2.2.1 Basic Values

We fix two disjoint finite sets $\mathbb{I}$ and $\mathbb{O}$, representing the input and output values, respectively. We also fix a special unknown value $\perp \notin \mathbb{I} \cup \mathbb{O}$. Given a set s not containing $\perp$, we write $s_\perp$ to denote $s \cup \{\perp\}$.

Informally, the $\perp$ value can represent three things. When provided as an input for a process, this represents that it does not participate in the protocol. Moreover, if a participating process crashes halfway through the execution, we take its output value to be $\perp$. Later, when we introduce the notion of shared-memory in subsection 2.2.4, $\perp$ is the default value stored in every cell of the global memory.

We fix a finite number $n \in \mathbb{N}$ of processes, each labelled from 0 to $n-1$. We write $[n]$ for the set $\{0, 1, \ldots, n-1\}$.

Given a set s, an n-tuple $m \in s^n$, $i \in [n]$, and a value $x \in s$, we write $m[i]$ or m_i for the i-th component of m and $m[i \leftarrow x]$ for m with the i-th component replaced by x. In addition, if we have $p \subseteq [n]$, we write $m[p \leftarrow x]$ for m with each i-th component replaced by x for each $i \in p$. Finally given $m \in (\mathbb{N} \to s^n)$, we write $m[(j,i) \leftarrow x]$ as the same function m but with the j-th component replaced by $m(j)[i \leftarrow x]$ for all $j \in \mathbb{N}$.

2.2.2 Protocol

Intuitively, a **protocol** is a family of programs, where each individual program is executed by a distinct process. It should describe how the state of the overall system changes when a process **w**rites to or **r**eads from the main memory. Thus, we define a protocol as follows:

Definition 2.2.1 (Protocol)**.** A protocol is a tuple $(\mathbb{V}, \pi)$, where $\mathbb{V}$ is a superset of $\mathbb{I} \cup \mathbb{O} \cup \{\perp\}$, and π consists of two families of functions:

$$\pi_{w_i} : \mathbb{V} \times \mathbb{V} \to \mathbb{V} \text{ and } \pi_{r_i} : \mathbb{V} \times \mathbb{V}^n \to \mathbb{V}$$

indexed by $i \in [n]$, satisfying the following three conditions:

$$\begin{aligned}
\pi_{w_i}(x, y_1) &= \pi_{w_i}(x, y_2) && \forall x \in \mathbb{V} \setminus \mathbb{O}_\perp, \forall y_1, y_2 \in \mathbb{V}, \forall i \in [n] && \text{(Global Memory Irrelevance)} \\
\pi_{w_i}(x, y) &= y && \forall x \in \mathbb{O}_\perp, \forall y \in \mathbb{V}, \forall i \in [n] && \text{(Write After Commit)} \\
\pi_{r_i}(x, m) &= x && \forall x \in \mathbb{O}_\perp, \forall m \in \mathbb{V}^n, \forall i \in [n] && \text{(Read After Commit)}
\end{aligned}$$

We sometimes simply write π for the protocol if the underlying set $\mathbb{V}$ can be inferred easily.

Intuitively, π_{w_i} defines how the i-th process changes a certain global memory cell according to its local state. It takes in two arguments, the former being the value of the local state and the latter being the value of the memory cell it is writing into. Note that the Global Memory Irrelevance condition ensures that for a normal write, the result does not depend on the original value the memory cell stores before

the write. This creates the desired effect that the write only depends on the process' local memory and nothing else (there are no hidden reads performed). When the local value is in $\mathbb{O}_\perp$, the condition Write After Commit creates the desired effect that any subsequent write does not change the value of the memory cell, and thus the entire system. This allows us to capture the notion that the process has committed, and does not perform any extra actions to change the system's state, including its own.

On the other hand, π_{r_i} defines how the i-th process changes its local state according to its original local state and an entire memory array block. It takes in two arguments, the former being its local state, and the latter being the values of the memory array. Similarly to Write After Commit, the condition Read After Commit ensures that once the process commits, it cannot subsequently change its local state.

2.2.3 Traces

An execution trace describes a possible interleaving of actions performed by a system of concurrent processes. The actions we are interested in for a trace are reads r, writes w, and crashes d.

We write the actions of the i-th process $\mathbb{A}_i = \{r_i, w_i, d_i\}$ and actions of all processes $\mathbb{A} = \bigcup_{i\in[n]} \mathbb{A}_i$.

We define $\mathbb{A}^*$ as the free monoid over $\mathbb{A}$ and $\mathbb{A}^\omega$ as the set of countably infinite sequences of actions. We define the set of traces $\mathbb{A}^{*+\omega}$ as $\mathbb{A}^* \cup \mathbb{A}^\omega$. We write ϵ for the empty trace, and $T \cdot U$ or simply TU for the concatenation of two traces T and U (where T must be a finite trace).

We write $\text{proj}_i : \mathbb{A}^{*+\omega} \to \mathbb{A}_i^{*+\omega}$ for the projections, keeping only the actions of the i-th process. Similarly, we have $\text{proj}_{\neg i} : \mathbb{A}^{*+\omega} \to \mathbb{A}^{*+\omega}$ for the projection keeping all actions except those of the i-th process. *Mutatis mutandis*, for all $s \subseteq [n]$, we have proj_s and $\text{proj}_{\neg s}$ with the obvious meaning. Given trace T, we write $\text{dead}(T)$ for the set of indices $i \in [n]$ where d_i occurs in T.

We may rewrite an execution trace into a numbered trace where all actions are decorated with the number of reads appearing before, e.g. we write a^p if there are p occurrences of r_i in the trace beforehand for any $a \in \mathbb{A}_i$.

Since any non-numbered trace has a corresponding numbered trace and vice versa, we interchange between using a normal trace and a numbered one. For example, we consider the finite non-numbered trace $r_0 r_2 w_0 w_1 r_1 r_1$ to be the same as the numbered trace $r_0^0 r_2^0 w_0^1 w_1^0 r_1^0 r_1^1$, and vice versa.

Given finite trace T and possibly infinite trace U, we say T is a prefix of U if there exists a trace T' such that $T \cdot T' = U$.

In an asynchronous distributed system, actions of each process interleave arbitrarily, and processes might crash at any point, leading to many possible execution traces. We use a **trace property** to define a set of possible execution traces we are interested in for a distributed system.

Definition 2.2.2 (Trace property)**.** A trace property Γ is a subset of $\mathbb{A}^{*+\omega}$.

This immediately begs the question: what should the trace property for our base model be? In addition to allowing arbitrary interleaving of actions, for any non-dead process, we also want them to have some sort of liveness property, i.e. they eventually execute a write (or a read) for any execution trace. Moreover, the trace can also contain d_i actions for any $i \in [n]$. Given these considerations, the trace property Γ_{Fair} for our base model is defined as follows:

Definition 2.2.3 (Fair traces)**.** A trace $T \in \mathbb{A}^{*+\omega}$ is fair if for all $i \in [n]$, $\text{proj}_i(T)$ either contains an infinite number of r_i and w_i actions, or contains a d_i action.

The trace property Γ_{Fair} contains all fair traces $T \in \mathbb{A}^{*+\omega}$.

2.2.4 Non-layered Model

In our framework, processes communicate with each other by reading and writing on some **shared memory**[1]. For our base model, we have a single n-sized global memory, where each process has write access to a distinct cell on the array. If a model uses this memory array as the communication primitive, we call this model a **non-layered model**.

Given a protocol and a finite trace, we can now define how the state of the system changes for a non-layered model as follows:

Definition 2.2.4 (Semantics of the non-layered model)**.** Given protocol $(\mathbb{V}, \pi)$ and a finite trace T, we define the semantics of the non-layered model $[\![T]\!]_\pi : \mathbb{V}^n \times \mathbb{V}^n \to \mathbb{V}^n \times \mathbb{V}^n$ inductively on the length of T:

$$\begin{aligned}
[\![w_i \cdot T]\!]_\pi(l, m) &= [\![T]\!]_\pi(l, m[i \leftarrow \pi_{w_i}(l_i, m_i)]) \\
[\![r_i \cdot T]\!]_\pi(l, m) &= [\![T]\!]_\pi(l[i \leftarrow \pi_{r_i}(l_i, m)], m) \\
[\![d_i \cdot T]\!]_\pi(l, m) &= [\![\mathrm{proj}_{\neg i}(T)]\!]_\pi(l, m) \\
[\![\epsilon]\!]_\pi(l, m) &= (l, m)
\end{aligned}$$

The definitions of this should not be surprising. For a w_i action, the i-th process writes to its assigned cell with the π_{w_i} function. For a r_i action, it reads the entire memory and updates its local state with the π_{r_i} action. If a d_i action is executed, all subsequent actions of the i-th process are removed, creating the effect that it crashed and thus does not have any external effects to the state of the system afterwards.

We use fst and snd to denote the first and second projections of a two element tuple, respectively. So $\mathrm{fst}[\![T]\!]_\pi(l, m)$ and $\mathrm{snd}[\![T]\!]_\pi(l, m)$ returns the final local state of the processes and the global memory after executing the trace T on base local state l and base global memory m, respectively.

We end this subsection by presenting three lemmas, which appear frequently in subsequent proofs.

Our first result is the Committed value lemma. Due to the Read After Commit condition, after a process commits, its local state remains unchanged as expected.

Lemma 2.2.1 (Committed value lemma)**.** For all protocols $(\mathbb{V}, \pi)$, vectors $l, m \in \mathbb{V}^n$, finite trace $T \in \mathbb{A}^{*+\omega}$, and $i \in [n]$, if we have $l[i] \in \mathbb{O}_\perp$, then $\mathrm{fst}([\![T]\!]_\pi(l, m))[i] = l[i]$.

Proof. Proved by strong induction on the length of T. See Appendix A.1 □

Our second result, the Constant local memory lemma states that the local memory of the i-th process remains unchanged if an execution trace does not contain the r_i action.

Lemma 2.2.2 (Constant local memory lemma)**.** For all finite traces T, protocols $(\mathbb{V}, \pi)$, $k \in \mathbb{N}$ and $i \in [n]$, if T does not contain r_i, for all $l, m \in \mathbb{V}^n$, $\mathrm{fst}([\![T]\!]_\pi(l, m))[i] = l[i]$.

Proof. Proved by strong induction on the length of T. □

Lastly, the Constant global memory lemma for non-layered models states that the i-th cell of the global memory remains unchanged if the trace does not contain the w_i action.

[1] It might seem strange that we use a shared-memory as the communication primitive for a distributed system instead of message-passing. This is because a shared-memory model is easier to reason about mathematically and is strong enough to simulate a message-passing model.

Lemma 2.2.3 (Constant global memory lemma for non-layered models)**.** For all finite traces T, protocols $(\mathbb{V}, \pi)$, and $i \in [n]$, if T does not contain w_i, for all $l, m \in \mathbb{V}^n$, $\text{snd}([\![T]\!]_\pi(l, m))[i] = m[i]$.

Proof. Proved by strong induction on the length of T. □

2.2.5 Task Description

We previously defined what a protocol and a trace are, and how they describe the change in state of a non-layered model. We now focus on the last aspect of the system: what it means for a model to solve a distributed task.

Given a set s containing $\perp$, we say two vectors $v, v' : s^n$ are a pair of matching vectors if for all $i \in [n]$, we have $v_i = \perp$ if and only if $v'_i = \perp$.

A task description is the distributed decision task we want our distributed system to solve, which we define as follows.

Definition 2.2.5 (Task description)**.** A task description is a relation $\Theta \subseteq \mathbb{I}^n_\perp \times \mathbb{O}^n_\perp$ such that for all $(l, l') \in \Theta$, l and l' are a pair of matching vectors[2].

This is not particularly complicated. A pair (l, l') in a task description is like a valid input-output pair of the overall distributed system. So if the initial system takes in l and returns l', we say the system solves the task. However, this is not enough since we only considered the case where no processes crash. We need to extend our definition of solving a task to include the possibility of dying processes. To provide additional flexibility on this definition, we do not fix a specific definition on task solvability, but allow the solvablity property to be any arbitrary predicate that depends on four data: the initial input, the final output, the set of nodes that crash during the trace, and of course, the task specification.

Definition 2.2.6 (Solvability property)**.** A solvability property P is a subset of $\mathbb{I}^n_\perp \times \mathbb{O}^n_\perp \times 2^{[n]} \times 2^{\mathbb{I}^n_\perp \times \mathbb{O}^n_\perp}$

Now, we tie everything up together by defining what it means for a model to solve a task description with respect to some solvability property:

Definition 2.2.7 (P-Solvable under the non-layered model)**.** We say a protocol π P-solves a task description Θ under the non-layered model for Γ traces if for all $l \in \mathbb{I}^n_\perp$ and all traces $T \in \Gamma$, there exists a finite prefix T' of T such that $P(l, \text{fst}([\![T']\!]_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$ holds.

The most interesting aspect of this definition is the part about choosing a prefix for a particular trace. This is because a trace might be infinite, and we can only reason about the semantics of a system under a finite execution trace. For any possibly infinite trace and input vector, we are interested in finding a certain prefix of the trace such that the input vector, the output vector (after executing the prefix trace) with the values of crashed processes masked with $\perp$, the set of crashed processes of the original trace, and the task description, satisfy the solvability property P.

We now present the Trace extension lemma. This useful lemma states that if we are able to find a prefix such that its semantics satisfies a solvability property, any other prefix that extends it also satisfies the same solvability property. The intuition behind this is that the Committed value lemma ensures that all committed and non-dead processes maintain the same local state.

[2] We write $x \in \Theta(y)$ to mean $(y, x) \in \Theta$.

Lemma 2.2.4 (Trace extension lemma)**.** For all distributed tasks Θ, protocols $(\mathbb{V}, \pi)$, $l \in \mathbb{V}^n$, traces T, solvability properties P, if $P(l, \text{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$ for some finite prefix T' of T, for all finite prefixes T'' of T satisfying $T' \leq T''$, then

$$P(l, \text{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$$

Proof. Proved by strong induction on the length of T. See Appendix A.2. □

Though all our propositions and subsequently, proofs, are all task solvability agnostic, we state the definition of GMT's and HKR's solvability property for completeness.

Definition 2.2.8 (GMT's solvability property [8])**.** We define $P_{\text{GMT}}(l, l', d, \Theta)$ to be true iff $l' \in \Theta(l[d \leftarrow \perp])$.

Definition 2.2.9 (HKR's solvability property [9])**.** We define $P_{\text{HKR}}(l, l', d, \Theta)$ to be true iff $\exists l'' \in \Theta(l)$ and that $l''[d \leftarrow \perp] = l'$

The two solvability properties mainly differ in how one considers the role of crashed processes. Recall that a task description only stores pairs of matching vectors. In GMT's version, we mask the original input vector with $\perp$ before determining whether the input-output pair is in the task description or not. In HKR's version, we check whether there exists some output vector, such that together with the input vector, they form a pair in the task description, and also that output vector, with the local state of crashed processes masked with $\perp$, is equivalent to the real output vector.

2.3 GMT's Model

We are now ready to precisely define GMT's model in the fundamental theorem. This is also the most general model we study in this report.

Definition 2.3.1 (GMT's model [8])**. GMT's model** is the **non-layered model**, where the possible execution traces are taken from $\Gamma_{\textbf{Fair}}$.

2.4 Discussion

We discuss how GMT's model, i.e. the non-layered model for Γ_{Fair} traces, aligns with the assumptions we listed in subsection 2.1.

1. We have $n \in \mathbb{N}$ processes in total, which is **finite**.
2. The sets of valid input and output values are fixed to be $\mathbb{I}$ and $\mathbb{O}$, respectively, which are fixed to be **finite**.
3. The non-layered model represents the shared-memory array from where processes write and read.

   A **read** action of the i-th process r_i updates its local state according to the state of the entire array and its original local state.

   A **write** action of the i-th process w_i updates the i-th cell of the memory array according to its local state. Though the i-th cell of the global memory is passed as a argument to the write function, the Global Memory Irrelevance condition ensures that the function does not depend on the value in the global memory.

4. A process participates in the protocol if it is given a valid input value from $\mathbb{I}$. It does not participate if it is given $\perp$ for the initial local state. In which case, it acts as if it has committed, so by the Write After Commit and Read After Commit conditions, it does not perform any actions on the state of the system even if it has an action in the execution trace. Note that the set of processes that receives $\perp$ as the initial state is **arbitrary**.

5. Execution traces from Γ_{Fair} might contain d_i actions which model the potential **crash** of a process. In which case, all future actions of the process are omitted when simulating the effect of the trace.

6. The trace property Γ_{Fair} allows arbitrary interleaving of actions by processes, thus giving us the **asynchronous** system property.

7. The **commit** action is assisted by the Write After Commit and the Read After Commit property as after a process changes its local state into a value in $\mathbb{O}_{\perp}$, its future actions does not affect the state of the overall system.

8. We have the **wait-free** property of processes since for all infinite trace from Γ_{Fair}, there must exists some finite prefix of the trace that satisfies the solvability property.

2.5 Preliminaries Summary

In this chapter, we presented the basic definitions of our framework, including the protocol, the trace, and the shared-memory. We defined the semantics of the non-layered model with respect to the above components and what it means for it to solve a distributed task with respect to a solvability property. Lastly, we defined GMT's model, and discussed how it reflects our original assumptions of a general asynchronous distributed system.

Chapter 3

Variants of the Base Model

In the previous chapter, we defined GMT's model which is our general base model. However, we also need to study other models that might have more interesting semantics and properties.

In this chapter, we study variations of our base models by changing various aspects of the system. In particular, we examine various trace properties, a different memory model, and other definitions of a protocol. We conclude this chapter by defining HKR's model, which is a combination of these variations.

3.1 Different Trace Properties

Recall that from subsection 2.2.3, our base model considers traces taken from the Γ_{Fair} trace property. We now consider smaller, more specific trace properties.

3.1.1 Valid Traces

Since subsequent actions of the i-th process are ignored after a d_i action (see subsection 2.2.4), it is natural for us to only consider traces which do not have any actions of the i-th process after a d_i action. We call these traces **valid**.

Definition 3.1.1 (Valid traces)**.** A valid trace T is a trace where for all $i \in [n]$, if $\text{proj}_i(T)$ contains a d_i action, it must be the last action in the projection. The trace property Γ_{valid} contains all valid traces $T \in \Gamma_{\text{fair}}$.

Remark. In GMT's work, a valid trace is called a strong properly dying trace.

An immediate benefit of valid traces is that their semantics is more elegant. In particular we have:

$$[\![d_i \cdot T]\!]_\pi(l, m) = [\![\text{proj}_{\neg i}(T)]\!]_\pi(l, m) = [\![T]\!]_\pi(l, m)$$

We can thus trivially prove the following nice compositionality property by induction.

Lemma 3.1.1 (Compositionality of valid traces)**.** For all finite valid traces T_1, T_2, and protocols π, we have

$$[\![T_1 T_2]\!]_\pi = [\![T_2]\!]_\pi \circ [\![T_1]\!]_\pi$$

Proof. Proved by induction on the length of T_1. □

We prove that restricting to Γ_{Valid} traces does not affect task solvability with respect to our base model in section 4.3.

3.1.2 Non-read-starting Traces

For each process, we might want to restrict its first action to be a write (given that it has not crashed). We call traces with this restriction **non-read-starting** traces. This new class of traces is used to define an intermediate model in the proof of the fundamental theorem.

Definition 3.1.2 (Non-read-starting traces)**.** A non-read-starting trace T is a trace where for all $i \in [n]$, the first action of $\text{proj}_i(T)$ is not r_i.

The trace property Γ_{Nrs} contains all non-read-starting traces $T \in \Gamma_{\text{Valid}}$.

We prove that restricting to Γ_{Nrs} traces does not affect task solvability with respect to our base model in section 4.4.

3.2 Alternating Traces

Note that in the previous trace properties, the exact sequence of actions executed by each process is quite arbitrary. We now introduce some order in how processes execute reads and writes, in particular, they must alternate between write and read actions. We call these traces **alternating**.

Definition 3.2.1 (Alternating traces)**.** A trace T is alternating if for each $i \in [n]$, $\text{proj}_i(T)$ is either a prefix of or exactly the infinite trace $w_i r_i w_i r_i \ldots$, or it is equals to $T' \cdot d_i$, where T' is a finite prefix of the above infinite trace.

The trace property Γ_{Alt} contains all alternating traces T such that for each $i \in [n]$, $\text{proj}_i(T)$ is exactly the infinite trace $w_i r_i w_i r_i \ldots$, or it is equals to $T' \cdot d_i$, where T' is a finite prefix of the above infinite trace.

We prove that restricting to Γ_{Alt} traces does not affect task solvability with respect to our base model in section 4.5.

3.3 Iterated Traces

In the previous defined trace properties, actions of various processes can be interleaved arbitrary, to capture the notion of asynchronicity. In more advanced models, we might have some sort of concurrency primitives, say a fence, that ensures that each process starts a new round of communication only if all (alive) processes finish executing the previous round. The trace can thus be split into intermediate segments, where in each segment, each process completes a round. We call these traces **iterated**.

Definition 3.3.1 (Iterated traces)**.** A numbered trace T is iterated if for all actions $a, b \in \{r, w, d\}$, if a_i^x, b_j^y are present in T and $x < y$, then a_i^x must appear before b_j^y in the trace.

The trace property Γ_{It} contains all iterated traces $T \in \Gamma_{\text{Alt}}$.

Restricting to Γ_{It} traces does **not** affect task solvability with respect to our base model. Instead, Γ_{It} is relevant when we consider the layered memory model, which we introduce in section 3.4.

3.3.1 Immediate Snapshot Traces

In our base model, we consider writes and reads to be two separate atomic actions. We now consider a slightly stronger communication primitive called an immediate snapshot. With an immediate snapshot, once a process executes a write, it then executes a read immediately. One should however **not** consider this write-read pair as a single atomic action. This is because a single immediate snapshot write-read pair can interleave with other processes' write-read pairs. By immediately, we mean that the actions can be re-ordered as if the write-read pairs of certain processes are executed **at the same time**. We call these traces **immediate snapshot traces**.

For example, the finite traces $w_0r_0w_1r_1w_2r_2$, $w_0w_1w_2r_1r_2r_0$, and $w_0r_0w_2w_1r_1r_2$ are all immediate snapshot traces. In particular, for the last trace, it has the effect of the 0-th process executing its write-read pair first, and the 1-st and 2-nd process executing their write-read pairs at the same time afterwards. As a non-example, the finite trace $w_0w_1r_1w_2r_2r_0$ is not immediate snapshot, since the write-read pair of the 0-th process is not immediate, spanning over the write-read pair of the 1-st and 2-nd processes, whose write-read pairs do not have the effect of happening in the same time.

Defining an immediate snapshot trace is not straightforward. We do so by first defining a function, updated:

Definition 3.3.2 (The updated function)**.** We define updated as the function that takes in a finite alternating trace T and returns a subset of $[n]$ defined by induction on the length of T as follows:

$$\begin{aligned}\text{updated}(\epsilon) &= \emptyset\\ \text{updated}(Tw_i) &= \text{updated}(T) \cup \{i\}\\ \text{updated}(Tr_i) &= \text{updated}(T) \setminus \{i\}\\ \text{updated}(Td_i) &= \text{updated}(T) \setminus \{i\}\end{aligned}$$

Definition 3.3.3 (Immediate snapshot traces)**.** A trace T is immediate snapshot if for every prefix of the form $T'r_iT''w_j$ where T'' only contains actions of the form d_k, $\text{updated}(T'r_iT'') = \emptyset$.

The trace property $\Gamma_{\text{It+IS}}$ contains all immediate snapshot traces $T \in \Gamma_{\text{It}}$.

Similarly to Γ_{It}, it is **not** the case that restricting to $\Gamma_{\text{It+IS}}$ traces does not affect task solvability with respect to our base model. The trace property $\Gamma_{\text{It+IS}}$ instead appears when we consider the layered memory model (see section 3.4).

Note the above trace properties form a total order ordered by set inclusion as follows:

$$\Gamma_{\text{It+IS}} \subseteq \Gamma_{\text{It}} \subseteq \Gamma_{\text{Alt}} \subseteq \Gamma_{\text{Nrs}} \subseteq \Gamma_{\text{Valid}} \subseteq \Gamma_{\text{Fair}}$$

This total order is useful because if we can solve a task decision for a certain trace property, we can solve the same task decision for a smaller subset of the trace property. This idea is introduced formally in section 4.2.

3.3.2 Bounded Traces

In all the above trace properties, the traces are all infinite (unless all processes crash in the trace). One may find it more natural to only consider traces where each process only executes a finite number of

steps. We call these traces **bounded**. We consider three variants of bounded trace properties, indexed by $k \in \mathbb{N}$ representing the number of write-read pairs each non-dying process performs.

Definition 3.3.4 (Bounded traces)**.** For all $k \in \mathbb{N}$, the trace property $\Gamma_{k\text{-Alt}}$ contains all alternating numbered traces T such that for each $i \in [n]$, $\text{proj}_i(T)$ is exactly the finite trace $w_i^0 r_i^0 w_i^1 r_i^1 \dots w_i^{k-1} r_i^{k-1}$, or it is equals to $T' \cdot d_i^m$ for some $m \in \mathbb{N}$, where T' is a finite prefix of the above finite trace.

For all $k \in \mathbb{N}$, the trace property $\Gamma_{k\text{-It}}$ contains all iterated traces $T \in \Gamma_{k\text{-Alt}}$.

For all $k \in \mathbb{N}$, the trace property $\Gamma_{k\text{-It+IS}}$ contains all immediate snapshot traces $T \in \Gamma_{k\text{-It}}$.

Similar to the total order we saw for non-bounded trace properties, for all $k \in \mathbb{N}$, we have the following total order for the bounded trace properties ordered by set inclusion:

$$\Gamma_{k\text{-It+IS}} \subseteq \Gamma_{k\text{-It}} \subseteq \Gamma_{k\text{-Alt}}$$

3.4 Layered Model

Recall that from subsection 2.2.4, we use a non-layered model, i.e. a single memory block for processes to read from and write onto. In HKR's model (see section 3.6), we use a more interesting memory model called the **layered model**. Instead of a single memory array, the layered model has an infinite number of memory arrays. For the i-th write-read pair of a process, it writes and reads from the i-th layer of the layered memory. This notion is captured formally by the following definition:

Definition 3.4.1 (Semantics of the layered model)**.** Given protocol $(\mathbb{V}, \pi)$ and a finite numbered trace T, we define the semantics of the layered model $[\![T]\!]_\pi : (\mathbb{V}^n \times (\mathbb{N} \to \mathbb{V}^n)) \to \mathbb{V}^n \times (\mathbb{N} \to \mathbb{V}^n)$ inductively on the length of T:

$$\begin{aligned}
[\![w_i^p \cdot T]\!]_\pi(l, m) &= [\![T]\!]_\pi(l, m[(p, i) \leftarrow \pi_{w_i}(l_i, m(p)[i])]) \\
[\![r_i^p \cdot T]\!]_\pi(l, m) &= [\![T]\!]_\pi(l[i \leftarrow \pi_{r_i}(l_i, m(p))], m) \\
[\![d_i^p \cdot T]\!]_\pi(l, m) &= [\![\text{proj}_{\neg i}(T)]\!]_\pi(l, m) \\
[\![\epsilon]\!]_\pi(l, m) &= (l, m)
\end{aligned}$$

The layered model is the reason why we might consider iterated traces. Intuitively, the label number of an action represents the layer a process writes into or reads from. Though we cannot have a memory model with an infinite number of layers in the physical world, restricting to iterated traces does not affect task solvability when using a layered model (see section 4.8).

The definition of task solvability for the layered model is similar to the non-layered one (see subsection 2.2.5):

Definition 3.4.2 (P-Solvable under the layered model)**.** We say a protocol π P-solves a task description Θ under the layered model for Γ traces if for all $l \in \mathbb{I}_\perp^n$ and all numbered traces $T \in \Gamma$, there exists a finite prefix T' of T such that $P(l, \text{fst}([\![T']\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$.

Though a different model, many results we have for the non-layered model still hold similarly for the layered one. We list them here below:

Lemma 3.4.1 (Committed value lemma for layered models)**.** For all protocols $(\mathbb{V}, \pi)$, $l \in \mathbb{V}^n$, $m \in \mathbb{N} \to \mathbb{V}^n$, finite numbered trace $T \in \mathbb{A}^{*+\omega}$, and $i \in [n]$, if we have $l[i] \in \mathbb{O}_\perp$, then $\text{fst}([\![T]\!]_\pi(l, m))[i] = l[i]$.

Proof. Similar to lemma 2.2.1. □

Lemma 3.4.2 (Constant local memory lemma for layered models)**.** For all finite traces T, protocols $(\mathbb{V}, \pi)$, $k \in \mathbb{N}$ and $i \in [n]$, if T does not contain r_i, for all $l \in \mathbb{V}^n, m \in \mathbb{N} \to \mathbb{V}^n$, $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, m))[i] = l[i]$.

Proof. Similar to lemma 2.2.2. □

Lemma 3.4.3 (Constant global memory lemma for layered models)**.** For all finite numbered traces T, protocols $(\mathbb{V}, \pi)$, $k \in \mathbb{N}$ and $i \in [n]$, if T does not contain w_i^k, for all $l \in \mathbb{V}^n, m \in \mathbb{N} \to \mathbb{V}^n$, $\mathrm{snd}(\llbracket T \rrbracket_\pi(l, m))[k][i] = m[k][i]$.

Proof. Similar to lemma 2.2.3. □

Lemma 3.4.4 (Trace extension lemma for layered models)**.** For all distributed tasks Θ, protocols $(\mathbb{V}, \pi)$, $l \in \mathbb{V}^n$, traces T, solvability properties P, if $P(l, \mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(T) \leftarrow \bot], \mathrm{dead}(T), \Theta)$ for some finite prefix T' of T, for all finite prefixes T'' of T satisfying $T' \leq T''$, then $P(l, \mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(T) \leftarrow \bot], \mathrm{dead}(T), \Theta)$.

Proof. Similar to lemma 2.2.4. □

Lemma 3.4.5 (Compositionality of valid traces for layered models)**.** For all finite valid numbered traces T_1, T_2 and protocols π, we have

$$\llbracket T_1 T_2 \rrbracket_\pi = \llbracket T_2 \rrbracket_\pi \circ \llbracket T_1 \rrbracket_\pi$$

Proof. Similar to lemma 3.1.1. □

3.5 Different Protocols

Recall that from subsection 2.2.2, a protocol consists of two families of functions for the write and read actions of processes. In this section, we consider variations of the protocol, by adding more conditions, or changing its type slightly.

3.5.1 Full-disclosure Protocols

Our first class of protocols is called the **full-disclosure** protocols. A non-committed process in a full-disclosure protocol writes the value of its local state directly into its global memory cell. This is captured by the following definition:

Definition 3.5.1 (Full-disclosure protocol)**.** A protocol $(\mathbb{V}, \pi)$ is a full-disclosure protocol if $\pi_{w_i}(x, y) = x$ for all $x \in \mathbb{V} \setminus \mathbb{O}_\bot$, $y \in \mathbb{V}$.

In fact, restricting to full-disclosure protocol does not affect task solvability with respect to the base model. We can see this from the following proposition:

Proposition 3.5.1 (Full-disclosure proposition)**.** For any distributed task Θ, trace property Γ, and solvability property P, Θ is P-solvable under the non-layered model for Γ traces if and only if there exists a full-disclosure protocol π that P-solves Θ under the non-layered model for Γ traces.

Proof. ($\Rightarrow$) Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ traces. We define the full-disclosure protocol $(\mathbb{V}, \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \begin{cases} x & \text{if } x \notin \mathbb{O}_\perp \\ y & \text{otherwise} \end{cases}$$
$$\pi'_{r_i}(x, m) = \pi_{r_i}(x, (\pi_{w_0}(m_0, \perp), ..., \pi_{w_{n-1}}(m_{n-1}, \perp)))$$

It is trivial to see that π' is a well-defined protocol. We claim that π' P-solves Θ under the non-layered model for Γ traces. The full proof can be found in Appendix B.

($\Leftarrow$) This direction is trivial because any full-disclosure protocol is also a protocol. □

The above result is used within one of the reduction proofs of the fundamental theorem (see section 4.4).

3.5.2 δ-protocols

We now define a different kind of protocol called **δ-protocols**[1]. Not only would this help us produce simpler, modular proofs for some algorithms, this is also an important component of HKR's model (see section 3.6).

Definition 3.5.2 (δ-protocol)**.** A δ-protocol is a tuple $(\mathbb{V}, \phi)$, where $\mathbb{V}$ is a superset of $\mathbb{I} \cup \mathbb{O} \cup \{\perp\}$, and ϕ consists of three families of functions:

$$\phi_{w_i} : \mathbb{V} \to \mathbb{V},\ \phi_{r_i} : \mathbb{V} \times \mathbb{V}^n \to \mathbb{V},\ \text{and}\ \phi_{\delta_i} : \mathbb{V} \to \mathbb{V}$$

indexed by $i \in [n]$, satisfying the following three conditions:

$$\phi_{w_i}(\perp) = \perp \qquad \forall i \in [n] \qquad \text{(Write strictness)}$$
$$\phi_{r_i}(\perp, m) = \perp \qquad \forall m \in \mathbb{V}^n, \forall i \in [n] \qquad \text{(Read strictness)}$$
$$\phi_{\delta_i}(\perp) = \perp \qquad \forall i \in [n] \qquad \text{(Decision strictness)}$$

The δ-protocol is similar to the normal protocol in that it still has the write and read functions for each process. One main difference is that the write function only depends on its local state, in which case, we can completely remove the Write After Commit condition. Instead, the conditions are replaced with strictness conditions. These are there to ensure that when a process is given $\perp$ as an input, we want it to not have any effect on the system as we consider it to be a non-participating process.

Definition 3.5.3 (Semantics of δ-protocols)**.** Given δ-protocol $(\mathbb{V}, \phi)$ and a finite trace T, its semantics in the non-layered and the layered model is the same as the normal protocol except for the write actions:

$$[\![w_i \cdot T]\!]_\phi(l, m) = [\![T]\!]_\phi(l, m[i \leftarrow \phi_{w_i}(l_i)])$$
$$[\![w_i^p \cdot T]\!]_\phi(l, m) = [\![T]\!]_\phi(l, m[(p, i) \leftarrow \phi_{w_i}(l_i)])$$

Given $l \in \mathbb{V}^n$, we write $\phi_\delta(l)$ as shorthand for $(\phi_{\delta_0}(l_0), \ldots, \phi_{\delta_{n-1}}(l_{n-1}))$.

[1] We explicitly state the term "δ-protocol" to refer to this special type of protocol. Otherwise, we use the term "protocol" to refer to the normal protocol in subsection 2.2.2.

Another difference of the δ-protocol is that it has an extra family of functions ϕ_{δ_i} indexed by $i \in [n]$. We call these decision functions. Informally, a distributed system running a δ-protocol has two stages. In the first stage, the processes **communicate** with each other via the shared memory, just like a normal protocol. In the second stage after exchanging information, the processes **decide** and commit by applying their decision function onto their local state. This notion is captured by the following definition of P-solvability for δ-protocols:

Definition 3.5.4 (P-Solvablity of δ-protocols)**.** We say a δ-protocol ϕ P-solves a decision task Θ under the non-layered model for Γ traces, where $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, if for all $l \in \mathbb{I}_\perp^n$ and all traces $T \in \Gamma$, we have $P(l, \phi_\delta(\text{fst}([\![T]\!]_\phi(l, \perp^n)))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$.

Similarly, ϕ solves a decision task Θ under the layered model for Γ traces, where $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, if for all $l \in \mathbb{I}_\perp^n$ and all traces $T \in \Gamma$, we have $P(l, \phi_\delta(\text{fst}([\![T]\!]_\phi(l, \lambda k.\perp^n)))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$.

Note that task solvability of δ-protocols only makes sense for bounded trace properties, because we want our first communication stage to be finite.

Though seemingly weaker than a normal protocol, we later show that a δ-protocol is as powerful as a normal protocol for finite, alternating traces. Both directions of this result are reduction steps in the proof of our fundamental theorem and can be found in section 4.9.

Note that despite having a different type and a set of different conditions, various results on the semantics of a normal protocol still hold for the δ-protocol, e.g. the Constant local memory lemma (see subsection 2.2.4).

3.5.3 Full-information δ-Protocols

We now consider a class of δ-protocols with extra conditions, called **full-information** δ-protocols. Similar to full-disclosure protocols (see subsection 3.5.1), in a full-information δ-protocol, processes write their entire local state into the global memory cell. In addition, during a read, processes condense the entire array block into a single value to store it as the local state. This notion is captured by the definition below:

Definition 3.5.5 (Full-information δ-protocol)**.** A δ-protocol $(\mathbb{V}, \phi)$ is a full-information δ-protocol if $\mathbb{V}$ is a set freely generated by a n-ary operator $(\_, \ldots, \_)$ on some superset of $\mathbb{I} \cup \mathbb{O}_\perp$ and the following two conditions are satisfied[2]:

$$\phi_{w_i}(x) = x \qquad \forall i \in [n], x \in \mathbb{V} \qquad \text{(Full-information write)}$$

$$\phi_{r_i}(x, m) = (m_0, \ldots, m_{n-1}) \qquad \forall i \in [n], x \in \mathbb{V} \setminus \{\perp\}, m \in \mathbb{V}^n \qquad \text{(Full-information read)}$$

We later show that if we can solve a task with a δ-protocol, we can always find a full-information δ-protocol to solve the same distributed task. This result is a component of the proof of the fundamental theorem and can be found in section 4.11.

[2] Note we are overloading the parenthesis () to represent two possible things: a vector of n $\mathbb{V}$ values, or a single $\mathbb{V}$ value of the n-ary tuple. Since they are isomorphic to each other, we refer to them interchangeably. A similar situation occurs in the proof of the Full-information δ-protocol proposition (see section 4.11) and the Non-layered Γ_{Alt} to layered Γ_{It} proposition (see section 4.13).

3.6 HKR's Model

After defining several variants of our original base model, we are now able to precisely describe HKR's model. HKR's model can be considered to be one of the most specific models in this report, and is used as the model in the Asynchronous Computability Theorem [10] (see subsection 5.2.3).

Definition 3.6.1 (HKR's model [9])**. HKR's model** is the **layered model**, where possible execution traces are taken from $\Gamma_{k\text{-}\mathbf{It+IS}}$ for some $k \in \mathbb{N}$, and executed by some **full-information** δ**-protocols**.

3.7 Variants of the Base Model Summary

In this chapter, we presented variants of the base model. We first studied various trace properties each with increasing number of conditions on its underlying traces. We then defined the layered memory model and discussed how it is different from the non-layered one. We also examined variations of the original protocols, either by adding more conditions or by slightly changing their definition. We finally conclude by defining HKR's model, which is considered to be the most specific model in the report.

Here, we successfully accomplished our first task, i.e. present a unified mathematical formal framework of asynchronous distributed systems, and define both GMT's and HKR's model. In the rest of the report, we focus on our second task, i.e. we now prove the Fundamental Theorem of Asynchronous Distributed Models, introduced in section 1.2.

Chapter 4

Fundamental Theorem

Now that we defined GMT's and HKR's models (see sections 2.3 and 3.6, respectively), we can unfold the definitions and present our fundamental theorem in full:

Theorem 4.0.1 (Fundamental Theorem of Asynchronous Distributed Models)**.** Given task description Θ and solvability property P, Θ is P-solvable under the non-layered model for Γ_{Fair} traces, iff there exists a full-information δ-protocol that P-solves Θ under the layered model for $\Gamma_{k\text{-It+IS}}$ traces for some $k \in \mathbb{N}$.

We start this chapter by providing an overview of the overall proof. Each subsequent section then contains a reduction algorithm from one model to another. This graph of models and reduction proofs containing both GMT's and HKR's model (see Figure 4.1), has a loop, and thus completes the proof[1].

4.1 Overview of the Proof

Before we present a high-level overview of the proof, we highlight aspects of the theorem statement that make the proof difficult.

1. Note that the theorem holds **for all** task descriptions Θ and solvability properties P. Consequently, the proofs must be agnostic to both variables.
2. Traces from GMT's model are taken from Γ_{Fair}, an unbounded trace property. Traces from HKR's model are instead taken from $\Gamma_{k\text{-It+IS}}$ for some $k \in \mathbb{N}$ **fixed in advance**, a bounded trace property. It is not obvious as to why solving a task with an unbounded trace property implies we can solve it with a bounded one, or vice versa.
3. GMT's model is a **non-layered** one and HKR's model is a **layered** one. It is not obvious as to why one model can be considered as strong as the other.
4. Traces from GMT's model are not necessarily **immediate snapshot**. Traces from HKR's model are all **immediate snapshot**. It is not trivial to prove that traces with the immediate snapshot property are as powerful as non-immediate snapshot ones.

With these difficulties in mind, we now present an overview of the proof of the Fundamental Theorem of Asynchronous Distributed Models:

[1] We encourage our readers to pause here, go away, and try proving the theorem on their own before reading on :)

Proof outline. The reduction steps from GMT's model to HKR's model, and vice versa, are presented in Figure 4.1, where each arrow is labelled with the corresponding section number of the proof. An arrow from model $\mathbb{A}$ to model $\mathbb{B}$ means that for any distributed task Θ and solvability property P, if model $\mathbb{A}$ can P-solve Θ, then so can model $\mathbb{B}$.

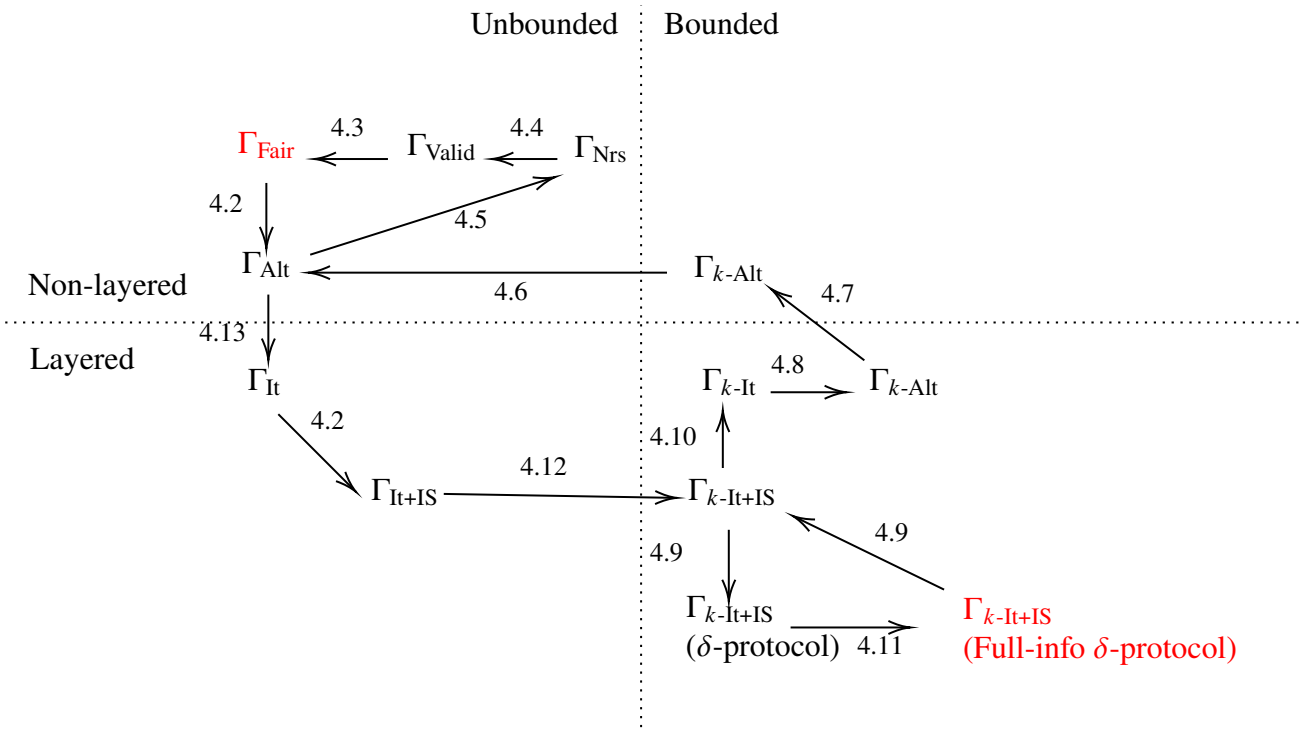


Figure 4.1: Proof of the Fundamental Theorem of Asynchronous Distributed Models

□

Remark. In GMT's paper, there exists the notion of a well-bracketed trace, which is similar to an alternating trace. In Appendix F, we study how alternating traces are similar to well-bracketed traces.

Here, we would like to highlight the fact that most of the above reduction steps follow a similar proof structure. Suppose we want to prove that if model $\mathbb{A}$ P-solves Θ, then model $\mathbb{B}$ also P-solves Θ. A reduction proof of that normally goes as follows:

1. Assume a (δ-)protocol α for model $\mathbb{A}$ that P-solves Θ.
2. Construct a (δ-)protocol β for model $\mathbb{B}$ from α, which we claim P-solves Θ with model $\mathbb{B}$.
3. Consider an arbitrary trace execution $T_{\mathbb{B}}$ allowed for model $\mathbb{B}$.
4. According to $T_{\mathbb{B}}$, find a trace execution $T_{\mathbb{A}}$ allowed for model $\mathbb{A}$.
5. By assumption, there exists some prefix trace $T'_{\mathbb{A}}$ of $T_{\mathbb{A}}$ such that the semantics of α for model $\mathbb{A}$ after executing $T'_{\mathbb{A}}$ satisfies P.
6. Show that this means that there exists some prefix trace $T'_{\mathbb{B}}$ of $T_{\mathbb{B}}$ such that the semantics of β for model $\mathbb{B}$ after executing $T'_{\mathbb{B}}$ satisfies P.

Most reduction proofs presented follow this structure, possibly with some slight deviations. The one reduction step that does not follow this style is that of the Layered $\Gamma_{\text{It+IS}}$ to layered $\Gamma_{k\text{-It+IS}}$ proposition (see section 4.12), where we use an interesting non-constructive proof by contradiction.

4.2 Trace Subset Proposition

We show that if we can solve a distributed task for some trace property, then we can also solve the same distributed task for a subset of that trace property.

We present this relatively trivial proposition here for two reasons. Firstly, this is the proof of two reduction steps, namely, from the non-layered model for Γ_{Fair} traces to the non-layered model for Γ_{Alt} traces, and from the layered model for Γ_{It} traces to the layered model for $\Gamma_{\text{It+IS}}$. Secondly, this proof presents a neat example of the proof outline highlighted in section 4.1 (where we choose $\beta = \alpha$ and $T_{\mathbb{A}} = T_{\mathbb{B}}$).

Proposition 4.2.1 (Trace subset proposition)**.** Given task description Θ, solvability property P, and trace properties Γ, Γ' with $\Gamma' \subseteq \Gamma$, if Θ is P-solvable under the non-layered model for Γ traces, then Θ is also P-solvable under the non-layered model for Γ' traces. This proposition also holds if we replace the non-layered model with a layered one.

Proof. By assumption, there exists a protocol π that P-solves Θ under the non-layered model for Γ traces. We claim that π also P-solves Θ under the non-layered model for Γ' traces.

Consider arbitrary $l \in \mathbb{I}^n_\perp$ and trace $T \in \Gamma'$. Since $\Gamma \subseteq \Gamma'$, we also have $T \in \Gamma$. By definition of P-solvability, there exists a finite prefix T' of T such that

$$P(l, \text{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$$

The proof for the layered model is similar. □

4.3 Non-layered Γ_{Valid} to Non-layered Γ_{Fair} Proposition

We show that solving a distributed task with the non-layered model for Γ_{Valid} implies that we can solve the same task with the non-layered model for Γ_{Fair} traces.

This proposition should not come as a surprise. Once a process dies, any of its subsequent actions in the trace are removed and it has no subsequent effect.

The difficulty in this proof lies in its constructive nature, where we have to precisely give an execution point in the fair trace that relates to the one we found in the valid trace which solves the task.

Proposition 4.3.1 (Non-layered Γ_{Valid} to non-layered Γ_{Fair} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Valid} traces, then Θ is also P-solvable under the non-layered model for Γ_{Fair} traces.

Proof. By assumption, there exists a protocol π that P-solves Θ under the non-layered model for Γ_{Valid} traces. We claim that π also P-solves Θ under the non-layered model for Γ_{Fair} traces.

Consider the following rewriting system with a family of rules $(R_i)_i$ for each $i \in [n]$:

$$R_i : T_a d_i T_b \Rightarrow T_a d_i \text{proj}_{\neg i}(T_b) \qquad \text{where } T_a \text{ does not contain } d_i$$

Lemma 4.3.2. The above rewriting system is terminating on all traces.

This is obvious since the rule R_i removes all actions of the i-th process after the first d_i actions if it exists, we can only apply it once. Each R_i also affects its own i-projection.

Lemma 4.3.3. For all $i \in [n]$, the rewriting system only containing R_i is confluent on all traces.

This is obvious since we can perform the rewrite on at most one place in the trace. If there are multiple d_i actions, we can only apply it in the first one.

Let f be the function which takes in a trace, and returns the trace after repeatedly applying R_i until it is irreducible in order of i.

Lemma 4.3.4. For all traces T, $\text{dead}(f(T)) = \text{dead}(T)$

This is obvious because we only remove actions in the rewriting rules, and the number of d_i actions never go to zero if there is at least one in the trace.

Lemma 4.3.5. For all traces $T \in \Gamma_{\text{Fair}}$, $f(T) \in \Gamma_{\text{Valid}}$.

Note first that after rewriting, $f(T)$ must be in Γ_{Fair}, by induction on number of rewrites. By contradiction if $f(T)$ is not valid, we would thus be able to perform another rewrite thus contradiction the fact that $f(T)$ is irreducible.

Lemma 4.3.6. For all finite prefixes T' of T, $[\![T']\!]_\pi = [\![f(T')]\!]_\pi$.

This can be proven by strong induction on the length of T'.

Lemma 4.3.7. For all $T, T' \in \Gamma_{\text{Fair}}$, if $T \leq T'$, we have $f(T) \leq f(T')$.

This can be shown by observing that if we can apply R_i on T, we can also apply it on T' and we must have $R_i(T) \leq R_i(T')$. Moreover if we can only apply R_i on T' but not T, we must have $T \leq R_i(T')$.

Lemma 4.3.8. For all countable increasing sequences $(T_i)_i$ under the prefix preorder $\leq$, where $\bigsqcup T_i = T$, we have $\bigsqcup_i f(T_i) = f(\bigsqcup_i T_i)$.

This can be shown with a similar argument as lemma 4.3.7.

Consider arbitrary $l \in \mathbb{I}_\perp^n$ and trace $T \in \Gamma_{\text{Fair}}$. We have $f(T) \in \Gamma_{\text{Valid}}$. By definition of P-solvability, there exists a prefix T' of $f(T)$ such that $P(l, \text{fst}([\![T']\!]_\pi(l, \perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta)$. Define T as $\bigsqcup_i T_i$ for some increasing sequence $(T_i)_i$, which is always possible. Since we have

$$
\begin{aligned}
& T' \\
\leq\ & f(T) && \text{(definition)} \\
=\ & f(\bigsqcup_i T_i) && \text{(definition)} \\
=\ & \bigsqcup_i f(T_i) && \text{(lemma 4.3.8)}
\end{aligned}
$$

there must exists some k such that $T' \leq f(T_k)$.

By the Trace extension lemma, we have $P(l, \text{fst}([\![f(T_k)]\!]_\pi(l, \perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta)$. By simple substitution, we also have $P(l, \text{fst}([\![T_k]\!]_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$, thus completing the proof.

□

4.4 Non-layered Γ_{Nrs} to Non-layered Γ_{Valid} Proposition

We show that solving a distributed task with the non-layered model for Γ_{Nrs} traces implies that we can solve the same task with the non-layered model for Γ_{Valid} traces.

The key idea is that our newly constructed protocol lets processes keep track on whether a write has been performed by themselves yet. If not, any read it performs does not change its local state. Note that this proof uses the Full-disclosure proposition (see subsection 3.5.1).

Proposition 4.4.1 (Non-layered Γ_{Nrs} to non-layered Γ_{Valid} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Nrs} traces, then Θ is also P-solvable under the non-layered model for Γ_{Valid} traces.

Proof. Suppose Θ is P-solvable under the non-layered model for Γ_{Nrs} traces. By the Full-disclosure proposition (see subsection 3.5.1), there exists a full-disclosure protocol $(\mathbb{V}, \pi)$ that P-solves Θ under the non-layered model for Γ_{Nrs} traces.

We construct a protocol $(\mathbb{V}, \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \pi_{w_i}(x, y)$$
$$\pi'_{r_i}(x, m) = \begin{cases} x & \text{if } x \in \mathbb{O}_\perp \text{ or } m_i = \perp \\ \pi_{r_i}(x, m) & \text{otherwise} \end{cases}$$

We claim that $(\mathbb{V}, \pi')$ P-solves Θ under the non-layered model for Γ_{Valid} traces. The full proof can be found in Appendix C.

□

4.5 Non-layered Γ_{Alt} to Non-layered Γ_{Nrs} Proposition

We show that solving a distributed task with the non-layered model for Γ_{Alt} traces implies that we can solve the same task with the non-layered model for Γ_{Nrs}.

This proposition is quite intuitive to understand. For each process, it does not provide additional information to others by writing multiple times consecutively. Similarly, it should not gain much by choosing to read multiple times consecutively. Thus, consecutive write/read actions of the same process can be compressed into one. The tediousness of the proof is a consequence of constructing the required protocol explicitly and proving its property.

Proposition 4.5.1 (Non-layered Γ_{Alt} to non-layered Γ_{Nrs} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Alt}, then Θ is also P-solvable under the non-layered model for Γ_{Nrs} traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ_{Alt} traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V} \cup \mathbb{N}$.

We define the function **ver**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{ver}(x) = \begin{cases} z & \text{if } x = \langle y, z \rangle \\ 0 & \text{otherwise} \end{cases}$$

We define the function **val**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} y & \text{if } x = \langle y, z \rangle \\ x & \text{otherwise} \end{cases}$$

We also define the function **map**: $((\mathbb{V}' \to \mathbb{V}') \times \mathbb{V}'^n) \to \mathbb{V}'^n$ as the function that applies the first argument to every element of the second argument, i.e. an n-tuple.

We define the protocol $(\mathbb{V}', \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\perp \\ \langle \pi_{w_i}(\mathbf{val}(x), \perp), \mathbf{ver}(x) \rangle & \text{otherwise} \end{cases}$$

$$\pi'_{r_i}(x, m) = \begin{cases} x & \text{if } x \in \mathbb{O}_\perp \\ x & \text{if } \mathbf{ver}(m[i]) \neq \mathbf{ver}(\mathrm{x}) \\ \begin{cases} \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)) & \text{if } \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)) \in \mathbb{O}_\perp \\ \langle \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)), & \text{otherwise} \\ \mathbf{val}(x) + 1 \rangle & \text{otherwise} \end{cases} \end{cases}$$

We claim that $(\mathbb{V}', \pi')$ P-solves Θ under the non-layered model for Γ_{Nrs} traces. The full proof can be found in Appendix D. □

4.6 Non-layered $\Gamma_{k\text{-Alt}}$ to Non-layered Γ_{Alt} Proposition

We show that solving a distributed task with the non-layered model for $\Gamma_{k\text{-Alt}}$ traces for some $k \in \mathbb{N}$ implies that we can solve the same task with the non-layered model for Γ_{Alt} traces.

The intuition behind this proof is that starting from any non-bounded alternating trace, we can repeatedly make use of the P-solvability definition to find a prefix of the trace where a process commits upon executing k rounds, so we can remove all subsequent actions of that process. Eventually we end up for a trace belonging to the bounded trace property. The main difficulty in this proof is defining explicitly how the bounded trace is constructed from the infinite trace.

Proposition 4.6.1 (Non-layered $\Gamma_{k\text{-Alt}}$ to non-layered Γ_{Alt} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for $\Gamma_{k\text{-Alt}}$ traces for some $k \in \mathbb{N}$, then Θ is also P-solvable under the non-layered model for Γ_{Alt} traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for $\Gamma_{k\text{-Alt}}$ traces for some $k \in \mathbb{N}$. We claim that $(\mathbb{V}, \pi)$ also P-solves Γ_{Alt}.

Consider arbitrary trace $T \in \Gamma_{\text{Alt}}$. Since $[n]$ is a finite set, one can find a prefix T' of T such that for all $i \in [n]$, either $w_i^0 r_i^0 \ldots w_i^{k-1} r_i^{k-1}$ is a prefix of $\text{proj}_i(T')$ or $\text{proj}_i(T') = T'' d_i$ where T'' is some prefix of $w_i^0 r_i^0 \ldots w_i^{k-1} r_i^{k-1}$. It suffices to prove that for all $l \in \mathbb{I}_\perp^n$, we have $P(l, \text{fst}([\![T']\!]_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$.

We now obtain a permutation $(p_i)_i$ of $0, \ldots, n-1$, such that the $r_{p_i}^{k-1}$ or d_{p_i} action (whichever is earlier) appears before the $r_{p_j}^{k-1}$ or d_{p_j} action in T' (whichever is earlier) for all $i < j$. Without loss of generality, we can let $p_i = i$.

We define a family of rewriting rules R_i indexed by $i \in [n]$:

$$R_i : T_a r_i^{k-1} T_b x_i T_c \Rightarrow T_a r_i^{k-1} T_b T_c \qquad \text{where } T_b \text{ does not contain } \mathbb{A}_i \text{ and } x_i \in \mathbb{A}_i$$

Lemma 4.6.2. For all $i \in [n]$, R_i is confluent and terminating for all finite traces.

We can see that R_i is confluent for all $i \in [n]$ because there is at most one place we can apply this rule. It is obviously terminating because applying R_i removes an action from the finite trace.

Let f_i be the function that takes in a finite trace and returns the trace that is rewritten by R_i if possible.

Lemma 4.6.3. For all $i \in [n+1]$, the trace $U = (f_{i-1} \circ \cdots \circ f_0)(T')$ has the following properties:

1. For all $j \in [i]$, $\text{fst}(\llbracket U \rrbracket_\pi(l, \bot^n))[j] = \text{fst}(\llbracket T' \rrbracket_\pi(l, \bot^n)[j]$
2. For all $j \in [i]$, $\text{proj}_j(U) = w_j^0 r_j^0 \dots w_j^{k-1} r_j^{k-1}$ or $U' d_j$ where U' is a prefix of $w_j^0 r_j^0 \dots w_j^{k-1} r_j^{k-1}$.
3. $\text{dead}(U) \subseteq \text{dead}(T)$

We prove this by induction on i.

- **Base case:** When $i = 0$, the first two conditions are vacuously true. The last condition is true because $\text{dead}(U) = \text{dead}(T') = \text{dead}(T)$
- **Inductive case:** Assume the statement is true for $i = q$ where $q \in [n]$. We prove that the statement is true for $i = q + 1$. Let $U' = (f_{q-1} \circ \cdots \circ f_0)(T')$ and $U = f_q(U')$.

  Note that the second condition is satisfied because for all $j \in [q]$, $\text{proj}_j(U) = \text{proj}_j(U')$ and if $j = q$, this condition is satisfied by case analysis on $\text{proj}_j(U')$.

  The third condition is also true because $\text{dead}(U) \subseteq \text{dead}(U') \subseteq \text{dead}(T)$.

  We now prove the first condition. Note that when $j \in [q]$, the statement follows by the Constant local memory lemma. It suffices to prove the case where $j = q$, which we do so by case analysis on whether R_q is applied. If no R_q is applied, $U' = U$ and we are done.

  Otherwise, let U_p be the prefix of U with the last element being the last action of q. Note that there must exists a trace $V \in \Gamma_{k\text{-Alt}}$ with U_p as a prefix and that $q \notin \text{dead}(V)$. This can be done because for all $j \in [q+1]$, $\text{proj}_j(U_p) = w_j^0 r_j^0 \dots w_j^{k-1} r_j^{k-1}$ or $U'_p d_j$ where U'_p is some prefix of $w_j^0 r_j^0 \dots w_j^{k-1} r_j^{k-1}$. For all $j \in [n] \setminus [q+1]$, $\text{proj}_j(U_p)$ must be a prefix of $w_j^0 r_j^0 \dots w_j^{k-1} r_j^{k-1}$. Thus, we can obtain V by appending d_j at the end of U_p for all $j \in [n] \setminus [q+1]$. Now by assumption, we must have some prefix V' of V such that $P(l, \text{fst}(\llbracket V' \rrbracket_\pi(l, \bot^n))[\text{dead}(V) \leftarrow \bot], \text{dead}(V), \Theta)$. By the Trace extension lemma, we have $P(l, \text{fst}(\llbracket V \rrbracket_\pi(l, \bot^n))[\text{dead}(V) \leftarrow \bot], \text{dead}(V), \Theta)$.

  Consider arbitrary $j \in [q+1]$. By inspecting the type of P, we thus have $\text{fst}(\llbracket V \rrbracket_\pi(l, \bot^n))[j] \in \mathbb{O}_\bot$. By the Constant local memory lemma, we have $\text{fst}(\llbracket U_p \rrbracket_\pi(l, \bot^n))[j] \in \mathbb{O}_\bot$. Finally, by the Committed value lemma, we have $\text{fst}(\llbracket U \rrbracket_\pi(l, \bot^n))[j] = \text{fst}(\llbracket U' \rrbracket_\pi(l, \bot^n))[j] = \text{fst}(\llbracket T' \rrbracket_\pi(l, \bot^n)[j]$.

From the above lemma, taking $i = n$, we have $U \in \Gamma_{k\text{-Alt}}$. We obtain U' by appending d_j for all $j \in \text{dead}(T) \setminus \text{dead}(U)$. Notice that U' is still in $\Gamma_{k\text{-Alt}}$ and $\text{dead}(U') = \text{dead}(T)$. It also remains the case that for all $i \in [n]$, we have $\llbracket U' \rrbracket_\pi = \llbracket U \rrbracket_\pi = \llbracket T' \rrbracket_\pi$. By assumption, we have $P(l, \text{fst}(\llbracket U' \rrbracket_\pi(l, \bot^n)[\text{dead}(U') \leftarrow \bot], \text{dead}(U'), \Theta)$. Performing required substitutions gives us $P(l, \text{fst}(\llbracket T' \rrbracket_(l, \bot^n)\pi)[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$, thus completing the proof.

□

4.7 Layered to Non-layered Proposition

We show that solving a distributed task with the layered model for most bounded alternating trace properties implies that we can solve the same task with the non-layered model for the same trace property.

It should not be surprising as to why the non-layered model can simulate the layered one; the layered model disallows processes from reading certain values written by others in previous rounds and intuitively seems more restrictive. In most papers, this proposition is often stated as obvious. The author still presents a proof of the proposition to illustrate how details can be hidden behind intuition.

This result seems to suggest that the non-layered model appears to be a stronger communication primitive than the layered model. However, we later show that the layered model can, surprisingly, simulate the non-layered model (see section 4.13).

Proposition 4.7.1 (Layered to non-layered proposition)**.** Given task description Θ and solvability property P, $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, if Θ is P-solvable under the layered model for Γ traces, then it is also P-solvable under the non-layered model for Γ traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$, and the variable length list operator $[\_, \ldots, \_]$ on $\mathbb{V}$.

We assume we have the append @ function on lists with the obvious semantics. We also define the function $\mathbf{len} : \mathbb{V}' \to \mathbb{N}$ which returns the length of a list, and 0 if the input is not a valid list. Similarly we have $x[i]$ or x_i as the i-th value of the list represented by $x \in \mathbb{V}'$, but $\perp$ if it is not a list or we access an index beyond its length.

We also define the function $f : \mathbb{V}'^n \times \mathbb{N} \to \mathbb{V}'^n$, where $f(m, i)$ maps the function $\lambda v.v[i]$ over m.

We define the protocol $(\mathbb{V}', \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\perp \\ [\pi_{w_i}(x, \perp)] & \text{if } x \in \mathbb{I} \\ l @ [\pi_{w_i}(z, \perp)] & \text{if } x = \langle l, z \rangle \\ \perp & \text{otherwise} \end{cases}$$

$$\pi'_{r_i}(x, m) = \begin{cases} x & & \text{if } x \in \mathbb{O}_\perp \\ \begin{cases} \pi_{r_i}(z, f(m, \mathbf{len}(m_i) - 1)) \in \mathbb{O}_\perp & \text{if } \pi_{r_i}(z, f(m, \mathbf{len}(m_i) - 1)) \in \mathbb{O}_\perp \\ \langle m_i, \pi_{r_i}(z, f(m, \mathbf{len}(m_i) - 1)) \rangle & \text{otherwise} \end{cases} & \text{if } x = \langle l, z \rangle \\ \begin{cases} \pi_{r_i}(x, f(m, \mathbf{len}(m_i) - 1)) & \text{if } \pi_{r_i}(x, f(m, \mathbf{len}(m_i) - 1)) \in \mathbb{O}_\perp \\ \langle m_i, \pi_{r_i}(x, f(m, \mathbf{len}(m_i) - 1)) \rangle & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

We claim that this protocol P-solves Θ under the non-layered model for Γ traces.

We do this by defining the following invariant $I \subseteq \mathbb{V}'^n \times (\mathbb{N} \to \mathbb{V}'^n) \times \mathbb{V}'^n \times \mathbb{V}'^n \times \mathbb{A}^*$. We say that $I(l, m, l', m', T)$ holds iff

- The two global memories agree, i.e. for all $s \in \mathbb{N}, i \in [n]$, $m(s)[i] = m'[i][s]$.
- For all $i \in [n]$, one of these cases hold:
  1. $l_i = l'_i$ and $l_i \in \mathbb{O}_\perp$

2. $d_i \in T$
3. $l_i = l_i'$, $l_i \in \mathbb{I}$, $\text{proj}_i(T) = \epsilon$, and for all $s \in \mathbb{N}$, $m(s)[i] = \perp$.
4. $l_i = l_i'$, $l_i \in \mathbb{I}$, $\text{proj}_i(T) = w_i^0$, and $\textbf{len}(m_i') = 1$
5. $\langle z, l_i \rangle = l_i'$ for some $z \in \mathbb{V}'$, $l_i \notin \mathbb{O}_\perp$, last action of $\text{proj}_i(T)$ is a w_i^s for some $s \in \mathbb{N}$, and $\textbf{len}(m_i') = s + 1$
6. $\langle m_i', l_i \rangle = l_i'$, $l_i \notin \mathbb{O}_\perp$, last action of $\text{proj}_i(T)$ is a r_i^s for some $s \in \mathbb{N}$, and $\textbf{len}(m_i') = s + 1$

We now prove this lemma:

Lemma 4.7.2. For all traces $T \in \Gamma$, prefixes T' of T, $l_{\text{initial}} \in \mathbb{I}_\perp^n$, let $(l, m) = [\![T']\!]_\pi(l_{\text{initial}}, \lambda k.\perp^n)$ and $(l', m') = [\![T']\!]_{\pi'}(l_{\text{initial}}, \perp^n)$, we have $I(l, m, l', m', T')$.

We prove this by induction on the length of T'. Note that for the second condition, we only need to consider the node which executes an action at the end. Also note that T and consequently T' are valid traces, so we make use of the Compositionality of valid traces implicitly:

- $T' = \epsilon$**:** It is trivial to see that the two global memories agree with each other and case 3 holds.
- $T' = T''d_i$**:** It is trivial to see that two global memories still agree by the induction hypothesis, the Constant global memory lemma for layered models and the Constant global memory lemma for non-layered models. Case 2 holds obviously for i. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsections 2.2.4 and 3.4.1).
- $T' = T''w_i$**:** We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsections 2.2.4 and 3.4.1). For i, the cases we need to consider for the T'' prefix are cases 1, 3, and 6 since T is in $\Gamma \subseteq \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the write action, by the Write After Commit condition and Constant local memory lemma, both the local and global memories remain unchanged, and thus the two memories agree and case 1 holds.

  If case 3 is true, the write action must be w_i^0. Unfolding the definition of π' and considering the Constant local memory lemma shows that case 4 holds.

  If case 6 is true, the write action must be w_i^s, and the previous action of i must be w_i^{s-1} where $s \geq 1$. Unfolding the definitions and considering the Constant local memory lemma shows that case 4 holds.
- $T' = T''r_i$**:** We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsections 2.2.4 and 3.4.1). For i, the cases we need to consider for the T'' prefix are cases 1, 4, and 5 since T is in $\Gamma \subseteq \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the read action, by the Read After Commit condition, Constant global memory lemma for layered models, and Constant global memory lemma for non-layered models both the local and global memories remain unchanged, and thus the two memories agree and case 1 holds.

  If case 4 is true, the read action must be r_i^0. By the Constant global memory lemma for layered models and Constant global memory lemma for non-layered models, the memories must agree. We now perform another case split. If $\pi_{r_i}(l_i, m(0)) \in \mathbb{O}_\perp$, by unfolding the definitions, we have case 1. Otherwise, case 6 holds.

  If case 5 is true, the read action must be r_i^s, and the previous action of i must be r_i^s where $s \in \mathbb{N}$. By the Constant global memory lemma for layered models and Constant global memory lemma for non-layered models, the memories must agree. We now perform another case split. If $\pi_{r_i}(l_i, m(s)) \in \mathbb{O}_\perp$, by unfolding the definitions, we have case 1. Otherwise, case 6 holds.

Now consider arbitrary $T \in \Gamma$, $l \in \mathbb{I}^n_\perp$. By assumption and the Trace extension lemma, we have $P(l, \mathrm{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$.

It suffices now to prove $P(l, \mathrm{fst}(\llbracket T \rrbracket_{\pi'}(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$, which we do so by proving $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[\mathrm{dead}(T) \leftarrow \perp][i] = \mathrm{fst}(\llbracket T \rrbracket_{\pi'}(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp][i]$ for all $i \in [n]$.

- $i \in \mathbf{dead}(T)$: This case is obvious because LHS = $\perp$ = RHS.
- $i \notin \mathbf{dead}(T)$: It suffices to prove that $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[i] = \mathrm{fst}(\llbracket T \rrbracket_{\pi'}(l, \perp^n))[i]$. Note by inspecting the type of solvability property P, we must have $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[i] \in \mathbb{O}_\perp$. By lemma 4.7.2 and the fact that T is itself a prefix of T, we have $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[i] = \mathrm{fst}(\llbracket T \rrbracket_{\pi'}(l, \perp^n))[i]$ as required.

□

4.8 Layered $\Gamma_{k\text{-It}}$ to Layered $\Gamma_{k\text{-Alt}}$ Proposition

We show that solving a distributed task with the layered model for $\Gamma_{k\text{-It}}$ traces for some $k \in \mathbb{N}$ implies that we can solve the same task with the layered model for $\Gamma_{k\text{-Alt}}$ traces.

The intuition behind this is that for any execution trace, we can swap consecutive actions if they are acting on different layers. This is also one of the reasons why the layered model is easier to reason about, for it provides us with a nice rearranging property.

Proposition 4.8.1 (Layered $\Gamma_{k\text{-It}}$ to layered $\Gamma_{k\text{-Alt}}$ proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the layered model for $\Gamma_{k\text{-It}}$ traces for some k, then there exists $k' \in \mathbb{N}$ such that Θ is P-solvable under the layered model for $\Gamma_{k'\text{-Alt}}$ traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the layered model for $\Gamma_{k\text{-It}}$ traces for some $k \in \mathbb{N}$. We claim that $(\mathbb{V}, \pi)$ also P-solves the layered model for $\Gamma_{k\text{-Alt}}$ traces.

Consider arbitrary numbered trace $T \in \Gamma_{k\text{-Alt}}$. Now consider the following rewriting system with a family of rules $(R_{a_i^x, b_j^y})_{a_i^x, b_j^y}$ where $a, b \in \{r, w, d\}, i, j \in [n]$, and $x, y \in \mathbb{N}$.

$$R_{a_i^x, b_j^y} : T_1 a_i^x b_j^y T_2 \Rightarrow T_1 b_j^y a_i^x T_2 \qquad \text{where } x > y \text{ and } i \neq j$$

Lemma 4.8.2. The above rewriting system is terminating on all traces $T \in \Gamma_{k\text{-Alt}}$.

Notice firstly that whenever we apply a rewrite on a trace $T \in \Gamma_{k\text{-Alt}}$ to form T', T' is also in $\Gamma_{k\text{-Alt}}$ by definition.

Now suppose we replace each action in the trace by their layer number to form a new list l, then a rewrite decreases the number of inversions of l by exactly one. Since the minimum number of inversions by a list formed by a trace in $\Gamma_{k\text{-Alt}}$ is finite, i.e. 0 in the case of an iterated trace, the rewriting system must be terminating.

Let f by a function which takes in a trace in $\Gamma_{k\text{-Alt}}$, and returns the trace after repeatedly applying some rule in the rewriting system until it terminates[2].

[2] We have not proven that this rewriting system is confluent. This is not necessary for we only need to pick some terminating trace. Moreover, the confluence result is obvious.

Lemma 4.8.3. $f(T) \in \Gamma_{k\text{-It}}$.

We know that $f(T) \in \Gamma_{k\text{-Alt}}$. It suffices to prove that it is iterated. We prove this by contradiction. Suppose there exists a_i^x, b_j^y in $f(T)$ where $x < y$ and b_j^y appears before a_i^x. We also must have some $c_{k_1}^{z_1}, d_{k_2}^{z_2}$ where $z_1 < z_2$ and $d_{k_2}^{z_2}$ appears right before $c_{k_1}^{z_1}$ with no actions in between. This is because if this were not the case, the list with the projection of the action's layer would be monotonically increasing and we will not be able to find any inversion.

Now we must have $k_1 \neq k_2$ because $f(T)$ is alternating. However we can perform the rewrite rule $R_{d_{k_2}^{z_2}, c_{k_1}^{z_1}}$ which contradicts our assumption that no rewrite rule can be performed on $f(T)$ any further.

Lemma 4.8.4. For any protocol $(\mathbb{V}, \pi)$, we have $[\![T]\!]_\pi = [\![f(T)]\!]_\pi$.

This can be shown by induction on the number of rewriting steps performed on the trace. From the Compositionality of valid traces, we only need to prove $[\![a_i^x b_j^y]\!]_\pi = [\![b_j^y a_i^x]\!]_\pi$ for all $x > y$ and $i \neq j$. This is however obvious by unfolding the definitions and is left as an exercise for the readers.

Lemma 4.8.5. dead(T)=dead($f(T)$).

This is obvious by observing that each rewrite retains the same number of actions in the trace.

Consider arbitrary $l \in \mathbb{I}_\perp^n$. By our assumption and lemma 4.8.3, there exists a prefix T' of $f(T)$ such that $P(l, \text{fst}([\![T']\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta)$.

We then have:

$$
\begin{aligned}
& P(l, \text{fst}([\![T']\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta) \\
& \Rightarrow P(l, \text{fst}([\![f(T)]\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta) && \text{(Trace extension lemma)} \\
& \Rightarrow P(l, \text{fst}([\![T]\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta) && \text{(Lemma 4.8.4)} \\
& \Rightarrow P(l, \text{fst}([\![T]\!]_\pi(l, \lambda k.\perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta) && \text{(Lemma 4.8.5)}
\end{aligned}
$$

which completes our proof.

□

4.9 δ-protocol Proposition

We show that we can solve a distributed task with the non-layered model for most bounded alternating trace properties if and only if we can find a δ-protocol that solves the same task with the non-layered model for the same trace property. This result also holds for the layered model.

The intuition behind this is that the decision function of a δ protocol can easily be embedded in the final round of a normal protocol. Note however that extra care is needed to ensure that the newly constructed (δ-)protocol satisfies all the necessary conditions, e.g. Write After Commit.

Proposition 4.9.1 (δ-protocol proposition)**.** For any distributed task Θ, trace property $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, and solvability property P, Θ is P-solvable under the non-layered model for Γ traces if and only if there exists a δ-protocol ϕ that P-solves Θ under the non-layered model for Γ traces. This proposition is also true in the layered case.

Proof. We only prove this for the non-layered case. The layered case is similar.

($\Rightarrow$) : Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V}$.

We define the functions **val**, **his**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} a & \text{if } x = \langle a, b \rangle \\ x & \text{otherwise} \end{cases}$$
$$\mathbf{his}(x) = \begin{cases} b & \text{if } x = \langle a, b \rangle \\ \bot & \text{otherwise} \end{cases}$$

We define the δ-protocol $(\mathbb{V}', \phi)$ as follows:

$$\phi_{w_i}(x) = \pi_{w_i}(\mathbf{val}(x), \mathbf{his}(x))$$
$$\phi_{r_i}(x, m) = \begin{cases} \bot & \text{if } x = \bot \\ \langle \pi_{r_i}(\mathbf{val}(x), m), m \rangle & \text{otherwise} \end{cases}$$
$$\phi_{\delta_i}(x) = \mathbf{val}(x)$$

We claim that $(\mathbb{V}', \phi)$ P-solves Θ under the non-layered model for Γ traces. The full proof can be found in Appendix E.

($\Leftarrow$): Assume the δ-protocol $(\mathbb{V}, \phi)$ P-solves Θ under the non-layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V} \cup \mathbb{N}$.

We define the functions **val**, **ver**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} a & \text{if } x = \langle a, b \rangle \\ x & \text{otherwise} \end{cases}$$
$$\mathbf{ver}(x) = \begin{cases} b & \text{if } x = \langle a, b \rangle \\ 0 & \text{otherwise} \end{cases}$$

We define the normal protocol $(\mathbb{V}', \pi)$ as follows:

$$\pi_{w_i}(x, y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\bot \\ \phi_{w_i}(\mathbf{val}(x)) & \text{otherwise} \end{cases}$$
$$\pi_{r_i}(x, m) = \begin{cases} x & & \text{if } x \in \mathbb{O}_\bot \\ \begin{cases} \phi_{\delta_i}(\phi_{r_i}(\mathbf{val}(x), m)) & \mathbf{ver}(x)\text{=k-1} \\ \langle \phi_{r_i}(\mathbf{val}(x), m), \mathbf{ver}(x) + 1 \rangle & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

We claim that $(\mathbb{V}', \pi)$ P-solves Θ under the non-layered model for Γ traces. The full proof can be found in Appendix E.

□

4.10 Layered $\Gamma_{k\text{-It+IS}}$ to Layered $\Gamma_{k'\text{-It}}$ Proposition

We show that solving a distributed task with the layered model for $\Gamma_{k\text{-It+Is}}$ traces for some $k \in \mathbb{N}$ implies that we can solve the same distributed task with the layered model for $\Gamma_{k'\text{-It}}$ traces for some $k' \in \mathbb{N}$.

The algorithm in this proof is adapted from Borowsky and Gafni [1]. The intuition behind this algorithm is that we can use n layers of the layered memory to simulate a single immediate snapshot layer.

Before we proceed, we define another trace property that would aid our proof.

Definition 4.10.1 (Subset Trace Property)**.** Given set $s \subseteq [n]$ and trace property Γ, a trace T is in Γ^s if and only if $T = \text{proj}_s(T')$ for some $T' \in \Gamma$.

Informally, the superscript s is the set of processes participating in the trace. Thus we have $\Gamma^{[n]} = \Gamma$ for any trace property Γ.

Proposition 4.10.1 (Layered $\Gamma_{k\text{-It+IS}}$ to layered $\Gamma_{k'\text{-It}}$ proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the layered model for $\Gamma_{k\text{-It+IS}}$ traces for some $k \in \mathbb{N}$, then Θ is also P-solvable under the layered model for $\Gamma_{k'\text{-It}}$ traces for some $k' \in \mathbb{N}$.

Proof. Assume the protocol $(\mathbb{V}_{\text{base}}, \pi)$ P-solves Θ under the layered model for $\Gamma_{k\text{-It+IS}}$ traces. By the δ-protocol proposition (see section 4.9), we have a δ-protocol $(\mathbb{V}, \phi)$ that P-solves Θ under the layered model for $\Gamma_{k\text{-It+IS}}$ traces. Moreover, it suffices to find a δ-protocol that P-solves Θ under the layered model for $\Gamma_{k'\text{-It}}$ traces for some $k' \in \mathbb{N}$.

We define $\mathbb{V}'$ to be the set freely generated by the unary operator $\{\_\}$ and the trenary operator $\langle \_, \_, \_ \rangle$ on $\mathbb{V} \cup \mathbb{N}$.

We define **val**, **rnd**, and **bool** as functions defined by the respective projections for input values of the $\langle\rangle$ tuple, or returning $x, 0, 0$, respectively, when given x as input otherwise.

We define the function **cnt**: $\mathbb{V}'^n \to \mathbb{N}$ as follows:

$$\textbf{cnt}(m) = |\{i \in [n] \mid \exists y \in \mathbb{V}'.m[i] = \{y\}\}|$$

We also define the function **snap**: $\mathbb{V}'^n \to \mathbb{V}'^n$ as follows:

$$\textbf{snap}(m)[i] = \begin{cases} y & \text{if m[i]=\{y\}} \\ m[i] & \text{otherwise} \end{cases}$$

We define a δ-protocol $(\mathbb{V}', \phi')$ as follows:

$$\phi'_{w_i}(x) = \begin{cases} \bot & \text{if } x = \bot \text{ or } \mathbf{bool}(x) = 1 \\ \{\phi_{w_i}(\mathbf{val}(x))\} & \text{otherwise} \end{cases}$$

$$\phi'_{r_i}(x, m) = \begin{cases} \bot & \text{if } x = \bot \\ \begin{cases} \begin{cases} \langle \mathbf{val}(x), \mathbf{rnd}(x) + 1, 0 \rangle & \text{if } \mathbf{bool}(x) = 1 \\ \langle \phi_{r_i}(\mathbf{val}(x), \mathbf{snap}(m)), & \text{if } \mathbf{rnd}(x)\%n = n - 1 \\ \mathbf{rnd}(x) + 1, 0 \rangle & \text{otherwise} \end{cases} \\ \begin{cases} \langle \phi_{r_i}(\mathbf{val}(x), \mathbf{snap}(m)), & \\ \mathbf{rnd}(x) + 1, 1 \rangle & \text{if } \mathbf{cnt}(m) \geq n - (\mathbf{rnd}(x)\%n) \text{ and } \mathbf{bool}(x) = 0 \\ \langle \mathbf{val}(x), \mathbf{rnd}(x) + 1, \mathbf{bool}(x) \rangle & \text{otherwise} \end{cases} \end{cases} & \text{otherwise} \end{cases}$$

$$\phi'_{\delta_i}(x) = \phi_{\delta_i}(\mathbf{val}(x))$$

We claim that $(\mathbb{V}', \phi')$ P-solves Θ for $\Gamma_{k'\text{-It}}$ traces where $k' = n \times k$.

We define the following invariant $I : \mathbb{V}^n \times \mathbb{V}'^m \times \mathbb{N} \times \mathcal{P}([n])$ where $I(l, l', p, s)$ holds iff

- For all $i \in s$, one of these cases is true:
  1. $l[i] = l'[i] = \bot$
  2. $l'[i] = \langle l[i], p \times n, 0 \rangle$
  3. $l[i] = l'[i] \in \mathbb{I}$ and $p = 0$

Consider the following lemma:

Lemma 4.10.2. For all $s \subseteq [n]$, $p \in \mathbb{N}$, trace $T \in \Gamma^s_{n\text{-It}}$, and vectors $l_1 : \mathbb{V}^n$, $l_2 : \mathbb{V}'^m$ satisfying $I(l_1, l_2, p, s)$, there exists a trace $T \in \Gamma^s_{1\text{-It+IS}}$ where $\text{dead}(T) = \text{dead}(T')$ such that when we let $l'_1 = \text{fst}(\llbracket T \rrbracket_\phi(l_1, \lambda k.\bot^n))$ and $l'_2 = \text{fst}(\llbracket T' \rrbracket_{\phi'}(l_2, \lambda k.\bot^n))$, we have $I(l'_1, l'_2, p + 1, s \setminus \text{dead}(T))$.

Before we prove this lemma, we show why this is helpful.

Consider arbitrary trace $T' \in \Gamma_{k'\text{-It}}$ and $l : \mathbb{I}^n_\bot$. Note we can split T' into $k + 1$ segments where each segment T'_j contains actions labelled $j \times n$ to $(j + 1) \times n - 1$, with T'_k only containing d actions.

By lemma 4.10.2, we are able to produce a trace $T = T_0 T_1 \ldots T_{k-1} T'_{k-1}$, where each T_j is each produced by the lemma. It is not difficult to show that $T \in \Gamma_{k\text{-It+IS}}$. Let $l_1 = \text{fst}(\llbracket T \rrbracket_\phi(l, \lambda k.\bot^n))$ and $l_2 = \text{fst}(\llbracket T' \rrbracket_{\phi'}(l, \lambda k.\bot^n))$. By the lemma and induction, we have $I(l_1, l_2, k, [n] \setminus \text{dead}(T))$. By assumption we have $P(l, \phi_\delta(l_1)[\text{dead}(T) \leftarrow \bot], \text{dead}(T) \leftarrow \bot, \Theta)$. By the definition of the I invariant, we must also have $P(l, \phi'_\delta(l_2)[\text{dead}(T') \leftarrow \bot], \text{dead}(T') \leftarrow \bot, \Theta)$ as required.

We now prove lemma 4.10.2. Consider arbitrary $s \subseteq [n]$, $p \in \mathbb{N}$, trace $T \in \Gamma^s_{n\text{-It}}$, and vectors $l_1 : \mathbb{V}^n$, $l_2 : \mathbb{V}'^m$ satisfying $I(l_1, l_2, p, s)$.

We define T_i as the prefix of T with all actions of round number smaller than i. We call a process pending if after executing some trace, it has not died and the **bool** value of its local value is 0.

Lemma 4.10.3. For all $k \in [n]$, after executing T_k, there are at most $n - k$ processes still pending.

The base case where $k = 0$ is trivial. Assume there are at most $n - k$ processes still pending after executing T_k. The interesting case is when there are exactly $n - k$ processes still pending. If there is any process dying in the new round, the statement is thus true. In the other case, if there are no processes dying in the new round, for all the processes still pending in the previous round, there must be one of them which performed its read the last. It thus must have read $n - k$ values which is larger or equal to $n - k$ and thus the **bool** value must be updated to 1. Thus at most $n - k - 1$ processes are pending in the next round.

With this lemma, each process that stops pending at layer k must perform the same simulated read since there are at most $n - k$ processes still pending after the previous layer, and a read is simulated in that layer if at least $n - k$ values are seen, which satsifies the immediate snapshot characteristics.

Let S_k be the set of processes that stops pending after executing T_k.

Let the elements of S_k be $a_0, \ldots a_{|S_k|-1}$. We define $W(S_k) = w_{a_0} \ldots w_{a_{|S_k|-1}}$. Let $R(S_k) = e_{a_0} \ldots e_{a_{|S_k|-1}}$, where $a_i = d_i$ if i dies in T and r_i otherwise.

The required trace T' is thus $W(S_n)R(S_n) \ldots W(S_1)R(S_1)$.

□

Remark. A similar (but definitely more complicated) proof can be presented for the non-layered version of the proposition. However (fortunately for the author), that proof is not necessary for proving the main theorem of this report.

4.11 Full-information δ-protocol Proposition

We show that we can solve a distributed task with a δ-protocol under the non-layered model for most bounded alternating trace properties if and only if we can solve the same task with a full-information δ-protocol under the non-layered model for the same trace property. This result holds for the layered model as well.

The trick behind the proof is that we can construct a simulation relation between the states of the system under the original δ-protocol and the newly constructed full-information δ-protocol, such that for each action step, the simulation relation holds between the two systems.

Proposition 4.11.1 (Full-information δ-protocol proposition)**.** For any distributed task Θ, trace property $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, and solvability property P, there exists a δ-protocol ϕ that P-solves Θ under the non-layered model for Γ traces if and only if there exists a full-information δ-protocol ϕ' that P-solves Θ under the non-layered model for Γ cases. This proposition is also true in the layered case.

Proof. We only prove this for the non-layered case. The layered case is similar.

($\Rightarrow$): Assume the δ-protocol $(\mathbb{V}, \phi)$ P-solves Θ under the non-layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the n-ary operator $(\_, \ldots, \_)$ on $\mathbb{V}$.

We also define a family of functions $f_i : \mathbb{V}' \to \mathbb{V}$, indexed by $i \in [n]$, inductively as follows:

$$f_i(x) = \begin{cases} x & \text{if } x \in \mathbb{V} \\ \phi_{r_i}(f_i(x_i), (\phi_{w_0} \circ f_0(x_0), \ldots, \phi_{w_{n-1}} \circ f_{n-1}(x_{n-1}))) & \text{if } x = (x_0, \ldots, x_{n-1}) \end{cases}$$

We define a full-information δ-protocol $(\mathbb{V}', \phi')$ as follows:

$$\begin{aligned}
\phi'_{w_i}(x) &= x \\
\phi'_{r_i}(x, m) &= \begin{cases} \bot & \text{if } x = \bot \\ (m) & \text{otherwise} \end{cases} \\
\phi'_{\delta_i}(x) &= \phi_{\delta_i}(f_i(x))
\end{aligned}$$

We claim that $(\mathbb{V}', \phi')$ P-solves Θ for Γ traces.

We define the following invariant $I \subseteq \mathbb{V}^n \times \mathbb{V}^n \times \mathbb{V}'^n \times \mathbb{V}'^n \times \mathbb{A}^*$ where $I(l, m, l', m', T)$ holds if for all $i \in [n]$

- $l[i] = f_i(l'[i])$
- $m[i] = \pi_{w_i} \circ f_i(m'[i])$
- If the last action of $\text{proj}_i(T)$ is a write, then $m'[i] = l'[i]$

Consider the following lemma:

Lemma 4.11.2. For all $T \in \Gamma$, for all prefixes T' of T, vectors $l_{\text{initial}} \in \mathbb{I}_\bot^n$, let $(l, m) = [\![T']\!]_\phi(l_{\text{initial}}, \bot^n)$, let $(l', m') = [\![T']\!]_{\phi'}(l_{\text{initial}}, \bot^n)$, we have $I(l, m, l', m', T')$.

We prove this by induction on the length of T'. Note that T and subsequently T' are valid traces, so we make use of the Compositionality of valid traces implicitly:

- $T' = \epsilon$**:** All three conditions hold trivially.
- $T' = T''d_i$**:** By the constant memory lemmas (see subsection 2.2.4) and the induction hypothesis, the three conditions hold.
- $T' = T''w_i$**:** We assume the induction hypothesis for the T'' prefix. By the Constant local memory lemma and the Constant global memory lemma for layered models, the local memory of all processes and global memory other than the i-th index remains unchanged. Thus the first condition is satisfied.

  We now check the second condition:

$$\begin{aligned}
&\text{LHS} \\
=&\phi_{w_i}(l[i]) && \text{(semantics of } \delta\text{-protocol)} \\
=&\phi_{w_i}(f_i(l'[i])) && \text{(induction hypothesis)} \\
=&\phi_{w_i}(f_i(\phi'_{w_i}(l'[i]))) && \text{(semantics of } \delta\text{-protocol)} \\
=&\text{RHS}
\end{aligned}$$

  Finally, for the third condition, since the last action of $\text{proj}_i(T')$ is a write, we also have:

$$\begin{aligned}
&\text{LHS} \\
=&\phi_{w_i}(l'[i]) && \text{(semantics of } \delta\text{-protocol)} \\
=&l'[i] && \text{(Full-information write)}
\end{aligned}$$

- $T' = T''r_i$**:** We assume the induction hypothesis for the T'' prefix. By the Constant local memory lemma and the Constant global memory lemma for layered models, the global memory and the local memory other than the i-th process remains unchanged. Thus the second condition is satisfied.

  The third condition is also vacuously true since the last action of $\text{proj}_i(T')$ is a read.

  It suffices to check only the first condition. We perform a case analysis:

  - $l'[i] = \perp$**:** By the induction hypothesis, we also have $l[i] = \perp$. After the read, by the (Read strictness) condition, the first condition still holds.
  - $l'[i] \neq \perp$**:** Since T' is alternating, we know that the last action of $\text{proj}_i(T'')$ must be a write. Thus we have:

$$
\begin{aligned}
&\text{LHS} \\
=&\phi_{r_i}(l[i], m) && \text{(semantics of } \delta\text{-protocol)} \\
=&\phi_{r_i}(f_i(l'[i]), m) && \text{(induction hypothesis)} \\
=&\phi_{r_i}(f_i(l'[i]), (\phi_{w_0} \circ f_0(m'[0]), \ldots, \phi_{w_{n-1}} \circ f_{n-1}(m'[n-1])))) && \text{(induction hypothesis)} \\
=&\phi_{r_i}(f_i(m'[i]), (\phi_{w_0} \circ f_0(m'[0]), \ldots, \phi_{w_{n-1}} \circ f_{n-1}(m'[n-1])))) && \text{(induction hypothesis)} \\
=&f_i((m'_0, \ldots, m'_{n-1})) && \text{(definition of } f_i\text{)} \\
=&f_i(\phi'_{r_i}(l'[i], m')) && \text{(definition of } \phi'\text{)} \\
=&\text{RHS}
\end{aligned}
$$

Consider arbitrary trace $T \in \Gamma$ and vector $l_{\text{initial}} \in \mathbb{I}_\perp^n$.

By assumption, we have $P(l_{\text{initial}}, \phi_\delta(\text{fst}(l_{\text{initial}}, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$. By lemma 4.11.2, we thus also have $P(l_{\text{initial}}, \phi'_\delta(\text{fst}(l_{\text{initial}}, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$ as required.

($\Leftarrow$)**:** This direction is trivial because any full-information δ-protocol is also a δ protocol. □

4.12 Layered $\Gamma_{\text{It+IS}}$ to Layered $\Gamma_{k\text{-It+IS}}$ Proposition

We show that solving a distributed task with the layered model for $\Gamma_{\text{It+IS}}$ traces implies that we can solve the same task with the layered model for $\Gamma_{k\text{-It+IS}}$ for some $k \in \mathbb{N}$.

This proposition is special in the sense that the proof does not follow the proof outline presented in section 4.1. Instead, we make use of König's lemma [5] to show that it cannot be the case for a protocol to both solve a distributed task under the layered model for $\Gamma_{\text{It+IS}}$ traces while not being able to solve the same task under the layered model for $\Gamma_{k\text{-It+IS}}$ traces for all $k \in \mathbb{N}$.

Various papers claimed that this argument is obvious [2][6]. However, no paper ever provided a rigorous proof of this[3].

Proposition 4.12.1 (Layered $\Gamma_{\text{It+IS}}$ to layered $\Gamma_{k\text{-It+IS}}$ proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the layered model for $\Gamma_{\text{It+IS}}$ traces, then there exists $k \in \mathbb{N}$ such that Θ is P-solvable under the layered model for $\Gamma_{k\text{-It+IS}}$ traces.

[3] In fact, the author is not able to prove some variants of this proposition, such as the reduction from non-layered Γ_{Alt} to non-layered $\Gamma_{k\text{-Alt}}$ (even if we restrict our solvability property to be that of HKR's solvability property), even though they are claimed to be easily proven by König's lemma [2][6].

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the layered model for $\Gamma_{\text{It+IS}}$ traces. We claim that there exists $k \in \mathbb{N}$ such that $(\mathbb{V}, \pi)$ P-solves Θ under the layered model for $\Gamma_{k\text{-It+IS}}$ traces.

Assume the contrary, i.e. for all $k \in \mathbb{N}$, there exists $l \in \mathbb{I}^n_\perp$, $T \in \Gamma_{k\text{-It+IS}}$, such that there does not exist prefix T' of T such that $P(l, \text{fst}(\llbracket T' \rrbracket_\pi(l, \lambda k.\perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$. Since T is a prefix of itself, we cannot have $P(l, \text{fst}(\llbracket T \rrbracket_\pi(l, \lambda k.\perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$ for that chosen $k \in \mathbb{N}$, for all $l \in \mathbb{I}^n_\perp$ and $T \in \Gamma_{k\text{-It+IS}}$.

For all $i \in \mathbb{N}$, let f_i be the function that takes in an iterated numbered trace, and returns the finite prefix of the trace containing all actions numbered with an index smaller than i.

Lemma 4.12.2. For all $k, k' \in \mathbb{N}$ with $k \leq k'$, $T \in \Gamma_{k\text{-It+IS}}$, $f_{k'}(T) \in \Gamma_{k'\text{-It+IS}}$

This is trivial by checking the definition.

We now construct a graph G as follows:

- The set of vertices of G contains a distinguished root ☣ and tuples of the form (l, T, s) for all $l \in \mathbb{I}^n_\perp, k \in \mathbb{N}, T \in \Gamma_{k\text{-It+IS}}, s \subseteq [n]$.
- The edges of G come in two forms:
  - There is an edge between ☣ and (l, ϵ, s) for all $l \in \mathbb{I}^n_\perp, s \subseteq [n]$.
  - There is an edge between $(l, f_k(T), s)$ and (l, T, s) for all $l \in \mathbb{I}^n_\perp, k \in \mathbb{N}, T \in \Gamma_{(k+1)\text{-It+IS}}, s \subseteq [n]$ such that $\text{dead}(T) \subseteq s$ and $\neg P(l, \text{fst}(\llbracket f_k(T) \rrbracket_\pi(l, \lambda k.\perp^n))[s \leftarrow \perp], s, \Theta)$.

Note that G is not necessarily connected. Let H be the subgraph of G containing all vertices reachble from ☣.

Lemma 4.12.3. *H is a tree.*

Suppose all edges defined above are directed. Note each vertex has in-degree at most 1. Since H is connected, there does not exist any cycles in the graph and thus H must be a tree.

Lemma 4.12.4. *H is locally finite.*

This is trivial since the sets $\mathbb{I}^n_\perp, \mathcal{P}(n), \Gamma_{k\text{-It+IS}}$ is finite for all $k \in \mathbb{N}$.

Lemma 4.12.5. *H is infinite.*

It suffices to prove that for all $m \in \mathbb{N}$, there exists a path of length at least m from ☣. Consider arbitrary m.

By assumption, there exists $l \in \mathbb{I}^n_\perp, T \in \Gamma_{m\text{-It+IS}}$ such that

$$\neg P(l, \text{fst}(\llbracket T \rrbracket_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$$

By induction on m and the contrapositive of the Trace extension lemma, it can be shown that there exists a path of the form ☣, $(l, f_0(T), \text{dead}(T))$, $(l, f_1(T), \text{dead}(T)), \ldots, (l, f_m(T), \text{dead}(T))$, which is of length $m + 2 \geq m$.

We now cite a well-known result in graph theory.

Lemma 4.12.6 (König's lemma [5])**.** A locally finite tree is infinite if and only if it has an infinite path.

By König's lemma, there exists an infinite path in H. Since every vertex is reachable from ☣, there must exists an infinite path starting from ☣. It is not difficult to see that this path must be of the form ☣, $(l', T_0, s), (l', T_1, s), \ldots$ for some $l' \in \mathbb{I}^n_\bot$, $s \subseteq [n]$, $T_i \in \Gamma_{i\text{-It+IS}}$ for each $i \in \mathbb{N}$ such that $T_i \leq T_{i+1}$ for each $i \in \mathbb{N}$ and $\mathrm{dead}(\bigsqcup_i T_i) \subseteq s$.

Lemma 4.12.7. $\bigsqcup_i T_i \in \Gamma_{\text{It+IS}}$

This is obvious since each T_i is just an extension of the previous by adding an another round. The immediate snapshot property is also maintained.

By assumption, there must exists a prefix T' of $\bigsqcup_i T_i$ satisfying $P(l, \mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(\bigsqcup_i T_i) \leftarrow \bot], \mathrm{dead}(\bigsqcup_i T_i), \Theta)$. Consider T_j for some $j \in \mathbb{N}$ such that $T' \leq T_j$. By the Trace extension lemma, $P(l, \mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(\bigsqcup_i T_i) \leftarrow \bot], \mathrm{dead}(\bigsqcup_i T_i), \Theta)$.

By the property of our chosen path, we have $\neg P(l, \mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot], s, \Theta)$. It suffices to prove $P(l, \mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot], s, \Theta)$ to achieve a contradiction.

Lemma 4.12.8. There exists a trace $U \in \Gamma_{\text{It+IS}}$ such that $T_j \leq U$ and $\mathrm{dead}(U) = s$.

This is because we can define U as $T_j d_{i_0} d_{i_1} \ldots d_{i_q} U' U' U' \ldots$, where each (i_r) is a distinct element such that $\{i_0, \ldots, i_q\} = s \setminus \mathrm{dead}(\bigsqcup_i T_i)$ and $U' = w_{j_0} r_{j_0} w_{j_1} r_{j_1} \ldots w_{j_x} r_{j_x}$ where each (j_r) is a distinct element such that $\{j_0, \ldots, j_x\} = [n] \setminus s$. We can then check that U is iterated, immediate snapshot, and is in Γ_{alt}.

By assumption, there exists a prefix U'' of U such that $P(l, \mathrm{fst}(\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot], s, \Theta)$.

We perform a case split:

- $T_j \leq U''$**:** To achieve our contradiction, it suffices to prove that $\mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot] = \mathrm{fst}(\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot]$.

  Consider arbitrary $i \in [n]$, we perform a case analysis.

  - $i \in s$**:** We obviously have

    $$\begin{aligned}&\mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot][i]\\ &\quad= \bot\\ &\quad= \mathrm{fst}(\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot][i]\end{aligned}$$

  - $i \notin s$**:** We have

    $$\begin{aligned}&\mathrm{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot][i]\\ &\quad= \mathrm{fst}\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(\bigsqcup_i T_i) \leftarrow \bot][i] && (\mathrm{dead}(\bigsqcup_i T_i) \subseteq s)\\ &\quad\in \mathbb{O}_\bot && (\text{type of } P)\end{aligned}$$

    Since $\mathrm{dead}(\bigsqcup_i T_i) \subseteq s$, we have

    $$\mathrm{fst}\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[\mathrm{dead}(\bigsqcup_i T_i) \leftarrow \bot][i] = \mathrm{fst}\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[i]$$

    By the Committed value lemma and Compositionality of valid traces, we have

    $$\mathrm{fst}\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[i] = \mathrm{fst}\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[i]$$

    Finally with $\mathrm{dead}(\bigsqcup_i T_i) \subseteq s$, we have $\mathrm{fst}\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[i] = \mathrm{fst}\llbracket U'' \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot][i]$, which completes the proof.

- $U'' \leq T_j$**:** We have $P(l, \text{fst}(\llbracket T_j \rrbracket_\pi(l, \lambda k.\bot^n))[s \leftarrow \bot], s, \Theta)$ by the Trace extension lemma.

For each case a contradiction is achieved, thus completing our proof. □

Remark. We only prove this for the layered version, but we note that the proof also works for the non-layered version though it is not a helpful proposition for us.

4.13 Non-layered Γ_{Alt} to Layered Γ_{It} Proposition

We finally now come to the *pons asinorum* of this report. We show that solving a distributed task with the non-layered model for Γ_{Alt} traces implies that we can solve the same task with the layered model for Γ_{It} traces.

This proof is inspired by that from Gafni and Rajsbaum [7]. The key idea is the sequence of simulated reads (which we define in the proof later) recorded in the trace from Γ_{It} forms a relation with the reads performed by some trace in Γ_{Alt}.

Proposition 4.13.1 (Non-layered Γ_{Alt} to layered Γ_{It} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Alt} traces, then Θ is also P-solvable under the layered model for Γ_{It} traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ_{Alt} traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\{\_, \_\}$, the 4-ary operator $\langle \_, \_, \_, \_ \rangle$, and the n-ary operator $(\_, \ldots, \_)$ on $\mathbb{V} \cup \mathbb{N}$.

We define **v** and **c** as functions defined by the first and second projections of input values of the $\{\}$ tuple or returning $\bot$ or 0, respectively, instead.

We also define **val**, **arr**, **rnd**, **srnd** as functions defined by the first, second, and third projections of input values of the $\langle\rangle$ tuple, or returning x, $(\{\bot, 0\}, \ldots, \{\bot, 0\})$, 1, and 1, respectively, when given as input otherwise.

Given a vector $m : \mathbb{V}'^n$, we define the $| \_ |$ as follows:

$$| m | = \sum_{0 \leq i \leq n-1} \mathbf{c}(m_i)$$

Given a vector $m : \mathbb{V}'^n$ where the i-projection of m is of the form $(m[i][0], \ldots, m[i][n-1])$, we define the **top** and **clock** functions as follows:

$$\begin{aligned}
\mathbf{top}(m) &= (m'_0 \ldots m'_{n-1}) \\
&\quad \text{where } m'_i = \mathbf{v}(m[j][i]) \text{ for some } j \text{ such that } \mathbf{c}(m[j][i]) \geq \mathbf{c}(m[k][i]) \text{for all } k \\
\mathbf{clock}(m) &= (m'_0 \ldots m'_{n-1}) \\
&\quad \text{where } m'_i = \mathbf{c}(m[j][i]) \text{ for some } j \text{ such that } \mathbf{c}(m[j][i]) \geq \mathbf{c}(m[k][i]) \text{for all } k
\end{aligned}$$

We define a protocol $(\mathbb{V}', \pi')$ as follows:

$$\pi'_{w_i}(x,y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\perp \\ \mathbf{arr}(x)[i \leftarrow \{\pi_{w_i}(x,\perp), \mathbf{srnd}(\mathrm{x})\}] & \text{otherwise} \end{cases}$$

$$\pi'_{r_i}(x,m) = \begin{cases} x & \text{if } x \in \mathbb{O}_\perp \\ \begin{cases} \begin{cases} \pi_{r_i}(x, \mathbf{top}(m)) & \text{if } \pi_{r_i}(x, \mathbf{top}(m)) \in \mathbb{O}_\perp \\ \langle \pi_{r_i}(x, \mathbf{top}(m)), \mathbf{top}(m), & \mid \mathbf{top}(m) \mid = r \\ \mathbf{rnd}(x)+1, \mathbf{srnd}(m)+1 \rangle & \text{otherwise} \end{cases} & \text{otherwise} \\ \langle \mathbf{val}(x), \mathbf{top}(m), & \\ \mathbf{rnd}(x)+1, \mathbf{srnd}(x) \rangle & \text{otherwise} \end{cases} \end{cases}$$

We claim that $(\mathbb{V}', \pi')$ P-solves Θ under the layered model for Γ_{It} traces.

Consider arbitrary trace $T \in \Gamma_{\mathrm{It}}$. By the iterated property, T can be written as $T_0 T_1 T_2 \ldots$, where T_i contains all actions of round number i for $i \in \mathbb{N}$.

Lemma 4.13.2. For all $j \in \mathbb{N}$, $l_{\mathrm{input}} \in \mathbb{I}^n_\perp$, let $l = \mathrm{fst}[\![T_0 \ldots T_{j-1}]\!]_{\pi'}(l_{\mathrm{input}}, \lambda k.\perp^n)$, for each $i \in [n]$, one of the following cases is true:

1. $l_i \in \mathbb{O}_\perp$
2. $j = 0$ and $l_i \in \mathbb{I}$
3. $l_i = \langle a, b, j+1, c \rangle$ for some $a \in \mathbb{V}' \setminus \mathbb{O}_\perp$, $b = (b_0, \ldots b_{n-1}) \in \mathbb{V}'$, $c \in \mathbb{N}$ where $j \leq \mid b \mid$ and $\mathbf{c}(b_i) \leq c \leq \mathbf{c}(b_i) + 1$
4. $d_i \in T_0 \ldots T_{j-1}$

The above can be proven by induction on $j \in \mathbb{N}$.

When executing the trace segment T_j for $j \in \mathbb{N}$, for all non-dead process and non-committed processes, if during the read phase the condition $\mid \mathbf{top}(m) \mid$ is satisfied, we say it executes a **simulated read**.

Lemma 4.13.3. For all $j \in \mathbb{N}$, all processes that are not committed and are non-dead that executed a simulated read in the T_{j-1} round does so with the same $\mathbf{top}(m)$. Let $c = \mathbf{clock}(m)$ with that value of m. In addition, for all uncommitted processes not executing a simulated read in that T_{j-1} round, if we let $c' = \mathbf{clock}(m')$, we must have $c \leq c'$ when compared index-wise.

Let the clock vector **clock**(m) seen by each non-committed and non-dead process i for the round T_{j-1} by c_i. Depending on the order of reads in T_{j-1}, it is not difficult to see that the c_i's form a linear preorder when compared index-wise. Note a simulated read is only executed if $\mid c_i \mid = j$. At the same time we know that by lemma 4.13.2, we have $j \leq \mid c_i \mid$ for all $i \in [n]$. Thus only the processes reading the bottom element of the c_i preorder can execute a simulated write.

Lemma 4.13.4. For all $j \in \mathbb{N}$, if executing $T_0 \ldots T_{j-1}$, there exists a non-empty set of non-dead and non-committed processes, there exists $k \geq j$ such that, a process dies in T_k or executed a simulated read.

This follows easily from lemma 4.13.2. When no processes dies or executes a simulated read, the read memory clock $\mathbf{clock}(m)$ remains unchanged. As the **rnd** of the local memory each non-dead non-committed process increases after each T_j, eventually one will have to execute a simulated write.

For all $j \in \mathbb{N}$, $l_{\text{initial}} \in \mathbb{I}_\perp^n$, consider the trace $T_0 \dots T_{j-1}$, let the increasing vector clocks $\mathbf{clock}(m)$ of the executed simulated reads be $c_1 \dots c_k$ for some $k \in \mathbb{N}$. Let c_0 be the all 0 n-vector. For all $p \in [k+1]$, let R_p be the list of read actions for all processes executing a simulated read with the vector clock c_p, and W_p be a list of write actions for all processes i where $c_{p-1}[i] < c_p[i]$. We define the trace $T'_j = W_1 R_1 \dots W_k R_k$.

Let $l = \text{fst}[\![T_0 \dots T_{j-1}]\!]_{\pi'}(l_{\text{initial}}, \lambda k.\perp^n)$ and $l' = \text{fst}[\![T'_j]\!]_\pi(l_{\text{initial}}, \perp^n)$.

Lemma 4.13.5. For all $j \in \mathbb{N}$, $l_{\text{initial}} \in \mathbb{I}_\perp^n$, let $l = \text{fst}[\![T_0 \dots T_{j-1}]\!]_{\pi'}(l_{\text{initial}}, \lambda k.\perp^n)$ and $l' = \text{fst}[\![T'_j]\!]_\pi(l_{\text{initial}}, \perp^n)$, for all $i \in [n]$, one of the following cases is true:

1. $l'[i] = l[i] \in \mathbb{I} \cup \mathbb{O}_\perp$
2. $l[i] = \langle l'[i], a, b, c \rangle$ for some $a, b, c \in \mathbb{V}'$

This is not difficult to observe by induction.

Lemma 4.13.6. $\bigsqcup_j T'_j$ is alternating.

This is true because whenever a process i executes a simulated read twice consecutively, the i-th element of the **clock** it observes is increased by exactly one. Thus, there is only one read appearing between each write of the same process.

Lemma 4.13.7. $\bigsqcup_j T'_j$ is finite.

Assume the contrary, that $\bigsqcup_j T'_j$ is infinite. This means there exists some process i that executes an infinite number of simulated reads. For all process k in $\bigsqcup_j T'_j$ that do not die and do not appear infinitely, we add a d_k action behind the last k-th action appearing in the trace. Let this new trace be T_{new}. Note that T_{new} is in Γ_{Alt}. Moreover, i does not die in T_{new}. By assumption, there exists some prefix T_{prefix} of T_{new} such that $P(l_{\text{initial}}, \text{fst}[\![T_{\text{prefix}}]\!]_\pi(l_{\text{initial}}, \perp^n)[\text{dead}(T_{\text{new}}) \leftarrow \perp], \text{dead}(T_{\text{new}}), \Theta)$. It must thus be the case that $\text{fst}[\![T_{\text{prefix}}]\!]_\pi(l_{\text{initial}}, \perp^n)[i] \in \mathbb{O}_\perp$. By using the Constant local memory lemma and the Committed value lemma implicitly, one can easily find a $j' \in \mathbb{N}$ such that $\text{fst}[\![T'_{j'}]\!]_\pi(l_{\text{initial}}, \perp^n)[i] \in \mathbb{O}_\perp$. By lemma 4.13.5, we must have $\text{fst}[\![T_0 \dots T_{j'-1}]\!]_\pi(l_{\text{initial}}, \lambda k.\perp^n)[i] \in \mathbb{O}_\perp$. This is a contradiction, because process i cannot execute any more simulated reads after $T_0 \dots T_{j'-1}$ since it has already committed.

Lemma 4.13.8. For all $i \in [n]$, $\text{fst}[\![\bigsqcup_j T'_j]\!]_\pi(l_{\text{initial}}, \perp^n)[\text{dead}(T) \leftarrow \perp][i] \in \mathbb{O}_\perp$.

Since $\bigsqcup_j T'_j$ is finite, $\bigsqcup_j T'_j = T'_{j'}$ for some $j' \in \mathbb{N}$. Assume the contrary of the lemma, by lemma 4.13.5, it must be the case that there exists some processes where after executing $T_0 \dots T_{j'-1}$, it has not committed and will never die in trace T. However by lemma 4.13.4, one of those processes must execute another simulated read in future rounds, thus extending $\bigsqcup_j T'_j$ and thus a contradiction.

Let W be the trace containing write actions of all processes not in $\text{dead}(T)$. Let R be the trace containing read actions of all processes not in $\text{dead}(T)$. Let D be the trace containing dead actions of all process in $\text{dead}(T)$. Let $T_{\text{ult}} = (\bigsqcup_j T'_j)DWRWR \dots = T'_{j'}DWRWR \dots$. Note that $T_{\text{ult}} \in \Gamma_{\text{Alt}}$. By assumption, there exists a prefix T'_{ult} of T_{ult} satisfying $P(l_{\text{initial}}, \text{fst}[\![T'_{\text{ult}}]\!]_\pi(l_{\text{initial}}, \perp^n)[\text{dead}(T_{\text{ult}} \leftarrow \perp)], \text{dead}(T_{\text{ult}}), \Theta)$.

We now perform a case analysis:

- $T'_{\mathbf{ult}} \leq T'_{j'}$**:** By the Trace extension lemma and lemma 4.13.5, we have

  $$P(l_{\text{initial}}, \text{fst}[\![T_0 \dots T_{j'-1}]\!]_{\pi'}(l_{\text{initial}}, \lambda k.\perp^n)[\text{dead}(T \leftarrow \perp)], \text{dead}(T), \Theta)$$

  as required.

- $T'_{j'} \leq T'_{\mathbf{ult}}$**:** By the Committed value lemma and lemma 4.13.8, we have:

$$\mathrm{fst}[\![T'_{j'}]\!]_\pi(l_{\mathrm{initial}}, \bot^n)[\mathrm{dead}(T) \leftarrow \bot] = \mathrm{fst}[\![T'_{\mathrm{ult}}]\!]_\pi(l_{\mathrm{initial}}, \bot^n)[\mathrm{dead}(T) \leftarrow \bot]$$

  By lemma 4.13.5, we then similarly have

$$P(l_{\mathrm{initial}}, \mathrm{fst}[\![T_0 \ldots T_{j'-1}]\!]_{\pi'}(l_{\mathrm{initial}}, \lambda k.\bot^n)[\mathrm{dead}(T \leftarrow \bot)], \mathrm{dead}(T), \Theta)$$

  as required.

□

4.14 Fundamental Theorem Summary

Here, we accomplished the second task of this report, i.e. proving the Fundamental Theorem of Asynchronous Distributed Models. We first presented a bird's-eye view of our proof before rigorously proving each intermediate reduction. This theorem is the first result to show that the models studied by GMT and HKR are equivalent in solvability power, meaning that results related to solvability for one model can also be used to reason about the other, e.g. the Asynchronous Computability Theorem.

Chapter 5

Conclusion

5.1 Summary

Let us take a step back and admire our work.

This report presents **the first steps in creating a theory** unifying HKR's and GMT's work on asynchronous distributed systems. The flexibility of the framework allows us to reason about the task solvability of distributed tasks for different kinds of **protocols** (e.g. full-disclosure protocols, δ-protocols), **execution traces** generated by different communication primitives (e.g. immediate snapshot traces, iterated traces), and **task solvability properties** (e.g. P_{GMT} and P_{HKR}), all under the same system.

As a significant proof of concept, we proved the **Fundamental Theorem of Asynchronous Distributed Models**, a new result illustrating an equivalence in task solvability between one of GMT's most general distributed models, i.e. the non-layered model for possibly infinite Γ_{Fair} traces, and one of HKR's most specific and interesting distributed models, i.e. the layered model for finite $\Gamma_{k\text{-It+IS}}$ traces via a full-information δ-protocol. This theorem thus unites two separate lines of research, allowing us to share the results of task solvability of one model with the other. Importantly, the Asynchronous Computability Theorem can be extended to also capture the solvability power of GMT's model. Our fundamental theorem is **solvability property-agnostic**, meaning it holds true for both GMT's and HKR's solvability property, which makes the proofs more powerful, albeit significantly more complicated.

5.2 Future Work

We examine various directions and open questions for continuing the project.

5.2.1 GMT's Three Propositions

In GMT's paper, three propositions on task solvability are stated without proofs. The first proposition follows easily as a corollary of the Fundamental Theorem of Asynchronous Distributed Models in this report.

The second proposition is not true for all solvability properties P, as one can show that we can solve consensus with HKR's solvability property P_{HKR} (see subsection 2.2.5), which contradicts the FLP

result [4]. However, the author conjectures that the proposition is true at least for GMT's solvability property P_{GMT}.

Conjecture 1. Given task description Θ, Θ is P_{GMT}-solvable under the non-layered model for Γ_{Fair} traces iff Θ is P_{GMT}-solvable under the non-layered model for Γ_{It} traces.

The author conjectures that the third proposition is true for all solvability properties. To prove it, it suffices to show the following, which can be seen as the non-layered version of the Layered $\Gamma_{k\text{-It+IS}}$ to layered $\Gamma_{k'\text{-It}}$ proposition (see section 4.10):

Conjecture 2. Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model $\Gamma_{k\text{-It+IS}}$ traces for some $k \in \mathbb{N}$, then Θ is also P-solvable under the non-layered model for $\Gamma_{\text{k'-It}}$ traces for some $k' \in \mathbb{N}$.

The author believes that the proof for the Layered $\Gamma_{k\text{-It+IS}}$ to layered $\Gamma_{k'\text{-It}}$ proposition can be lifted to work for the non-layered version.

5.2.2 GMT's Protocol

The definition of a protocol in this report differs from GMT's definition slightly. In particular, GMT's definition of a protocol does not have the Write After Commit condition. Without this condition, some proofs are invalid, for example that of the Non-layered $\Gamma_{k\text{-Alt}}$ to non-layered Γ_{Alt} proposition (see section 4.6).

The author conjectures that even without the Write After Commit condition, the statement of all theorems is still true, though their proofs might require more sophisticated reasoning with the topological structure produced by the protocol.

Conjecture 3. The Fundamental Theorem of Asynchronous Distributed Models still holds if we consider protocols without the Write After Commit condition.

5.2.3 GMT's Solvability Property

It has been shown that distributed tasks that are P_{HKR}-solvable possess nice geometric properties, as illustrated in the Asynchronous Computability Theorem, which we present in the form of our framework:

Theorem 5.2.1 (Asynchronous Computability Theorem [10])**.** Given distributed task Θ, Θ is P_{HKR}-solvable under the non-layered model for Γ_{Fair} traces if and only if there exists a chromatic subdivision σ of the simplicial complex $\mathbb{I}^n_\perp$ and a color-preserving simplicial map $\mu : \sigma(\mathbb{I}^n_\perp) \to \mathbb{O}^n_\perp$, such that for each vertex $s \in \sigma(\mathbb{I}^n_\perp)$, $\mu(s) \in \Theta(s')$, where s' is the unique smallest simplex in $\mathbb{I}^n_\perp$ that contains s.

The author conjectures that one can find a similar (most likely simpler) geometric property to characterize tasks that are P_{GMT} solvable.

Conjecture 4. There exists a geometric characterization of all task descriptions P_{GMT}-solvable under the non-layered model for Γ_{Fair} traces.

5.2.4 The Category-theoretic Perspective of Task Solvability

Originally, the author aimed to examine whether category theory provides tools for us to understand or prove various theorems on task solvability, e.g. the Asynchronous Computability Theorem. The

project, however, deviated from this original plan when we realized that not enough work has been done to rigorously structure various results on distributed task solvability under a unified framework.

The author hopes to build upon their framework, and rewrite various proofs and results under the lens of category theory, so that the proofs can be more easily understood and generalized.

Bibliography

[1] E. Borowsky and E. Gafni. Immediate atomic snapshots and fast renaming. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC '93, page 41–51, New York, NY, USA, 1993. Association for Computing Machinery.

[2] E. Borowsky and E. Gafni. A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '97, page 189–198, New York, NY, USA, 1997. Association for Computing Machinery.

[3] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr 1985.

[5] M. Franchella. On the origins of Dénes König's infinity lemma. *Archive for History of Exact Sciences*, 51(1):3–27, 1997.

[6] E. Gafni, P. Kuznetsov, and C. Manolescu. A generalized asynchronous computability theorem, 2014.

[7] E. Gafni and S. Rajsbaum. Distributed programming with tasks. In C. Lu, T. Masuzawa, and M. Mosbah, editors, *Principles of Distributed Systems*, pages 205–218, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] E. Goubault, S. Mimram, and C. Tasson. Geometric and combinatorial views on asynchronous computability. *Distributed Computing*, 31(4):289–316, Aug. 2018.

[9] M. Herlihy, D. Kozlov, and S. Rajsbaum. Distributed computing through combinatorial topology. In *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, Boston, 2014.

[10] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, nov 1999.

[11] M. P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, page 276–290, New York, NY, USA, 1988. Association for Computing Machinery.

[12] M. Kleppmann. *Designing Data-Intensive Applications*. O'Reilly, Beijing, 2017.

Appendix A

Useful Lemmas

A.1 Committed Value Lemma

Lemma A.1.1 (Committed value lemma)**.** For all protocols $(\mathbb{V}, \pi)$, $l, m \in \mathbb{V}^n$, finite trace $T \in \mathbb{A}^{*+\omega}$, and $i \in [n]$, if we have $l[i] \in \mathbb{O}_\perp$, then $\mathrm{fst}(\llbracket T \rrbracket_\pi(l, m))[i] = l[i]$. We have the same result if we replace all $\llbracket \rrbracket$ with $\llbracket\llbracket \rrbracket\rrbracket$.

Proof. We prove by strong induction on length of T for all l, m.

- $T = \epsilon$**:** We then have $\llbracket T \rrbracket_\pi = \mathrm{id}$, and the result follows trivially.
- $T = w_j \cdot T'$**:** We have

$$\begin{aligned} & \mathrm{fst}(\llbracket w_j \cdot T \rrbracket_\pi(l, m))[i] \\ = {} & \mathrm{fst}(\llbracket T \rrbracket_\pi(l, m[j \leftarrow \pi_{w_j}(l_j, m_j)]))[i] && \text{(definition)} \\ = {} & l[i] && \text{(induction hypothesis)} \end{aligned}$$

- $T = r_i \cdot T'$**:** We have

$$\begin{aligned} & \mathrm{fst}(\llbracket r_i \cdot T \rrbracket_\pi(l, m))[i] \\ = {} & \mathrm{fst}(\llbracket T \rrbracket_\pi(l[i \leftarrow \pi_{r_i}(l_i, m)], m))[i] && \text{(definition)} \\ = {} & \mathrm{fst}(\llbracket T \rrbracket_\pi(l[i \leftarrow l_i], m))[i] && \text{(Read After Commit)} \\ = {} & l[i] && \text{(induction hypothesis)} \end{aligned}$$

- $T = r_j \cdot T'$ **where** $j \neq i$**:** We have

$$\begin{aligned} & \mathrm{fst}(\llbracket r_j \cdot T \rrbracket_\pi(l, m))[i] \\ = {} & \mathrm{fst}(\llbracket T \rrbracket_\pi(l[j \leftarrow \pi_{r_j}(l_j, m)], m))[i] && \text{(definition)} \\ = {} & l[i] && \text{(induction hypothesis)} \end{aligned}$$

- $T = d_j \cdot T'$**:** We have

$$\begin{aligned} & \mathrm{fst}(\llbracket d_j \cdot T \rrbracket_\pi(l, m))[i] \\ = {} & \mathrm{fst}(\llbracket \mathrm{proj}_{\neg j}(T) \rrbracket_\pi(l, m))[i] && \text{(definition)} \\ = {} & l[i] && \text{(induction hypothesis)} \end{aligned}$$

The proof is similar for the layered model. □

A.2 Trace Extension Lemma

Lemma A.2.1 (Trace extension lemma)**.** For all distributed tasks Θ, protocols $(\mathbb{V}, \pi)$, $l \in \mathbb{V}^n$, traces T, solvability properties P, if we have $P(l, \mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$ for some finite prefix T' of T, for all finite prefixes T'' of T satisfying $T' \leq T''$, we have:

$$P(l, \mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$$

We have a similar result for the layered model version.

Proof. It suffices to prove that $\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp] = \mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp]$. Let T''' be the trace such that $T' \cdot T''' = T''$.

Consider $i \in [n]$,

- **Case 1:** $i \in \mathbf{dead}(T)$**:** we must have

$$\begin{aligned} &\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp][i] \\ =\ &\perp \\ =\ &\mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp][i] \end{aligned}$$

- **Case 2:** $i \notin \mathbf{dead}(T)$**:** Note we must have $\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[i] \in \mathbb{O}_\perp$ by inspecting the type of P (see Definition 2.2.6).

  Also observe that for $\llbracket \mathrm{proj}_{\neg \mathrm{dead}(T')} T''' \rrbracket_\pi \circ \llbracket T' \rrbracket_\pi = \llbracket T'' \rrbracket_\pi$, which can be proven by induction.

$$\begin{aligned} &\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp][i] \\ =\ &\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n))[i] \\ =\ &\mathrm{fst}(\llbracket \mathrm{proj}_{\neg \mathrm{dead}(T')} T''' \rrbracket_\pi)(\mathrm{fst}(\llbracket T' \rrbracket_\pi(l, \perp^n)), \mathrm{snd}(\llbracket T' \rrbracket_\pi(l, \perp^n))[i] && \text{(Committed value lemma)} \\ =\ &\mathrm{fst}((\llbracket \mathrm{proj}_{\neg \mathrm{dead}(T')} T''' \rrbracket_\pi \circ \llbracket T' \rrbracket_\pi)(l, \perp^n))[i] \\ =\ &\mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[i] && \text{(By above property)} \\ =\ &\mathrm{fst}(\llbracket T'' \rrbracket_\pi(l, \perp^n))[\mathrm{dead}(T) \leftarrow \perp][i] \end{aligned}$$

The proof is similar for the layered model. □

Appendix B

Full Proof of the Full-disclosure Proposition

Proposition B.0.1 (Full-disclosure proposition)**.** For any distributed task Θ, trace property Γ, and solvability property P, Θ is P-solvable under the non-layered model for Γ traces if and only if there exists a full-disclosure protocol π that P-solves Θ under the non-layered model for Γ traces.

Proof. ($\Rightarrow$) Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ traces. We define the full-disclosure protocol $(\mathbb{V}, \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \begin{cases} x & \text{if } x \notin \mathbb{O}_\perp \\ y & \text{otherwise} \end{cases}$$
$$\pi'_{r_i}(x, m) = \pi_{r_i}(x, (\pi_{w_0}(m_0, \perp), ..., \pi_{w_{n-1}}(m_{n-1}, \perp)))$$

It is trivial to see that π' is a well-defined protocol.

Lemma B.0.2. For all $l, m, m' \in \mathbb{V}^n$, satisfying $m[i] = \pi_{w_i}(m'[i], \perp)$ for all $i \in [n]$, for all traces T, we have

$$\text{fst}([\![T]\!]_\pi(l, m)) = \text{fst}([\![T]\!]_{\pi'}(l, m'))$$

and for all $i \in [n]$

$$\text{snd}([\![T]\!]_\pi(l, m))[i] = \pi_{w_i}(\text{snd}([\![T]\!]_{\pi'}(l, m'))[i], \perp)$$

.

We prove this by strong induction on the length of T.

- $T = \epsilon$**:** We have $[\![T]\!]_\pi = [\![T]\!]_{\pi'} = \text{id}$, and the equations follow from assumptions.
- $T = w_i \cdot T'$**:** We have

$$\begin{aligned} &\text{fst}([\![w_i \cdot T']\!]_\pi(l, m)) \\ &\qquad = \text{fst}([\![T]\!]_\pi(l, m)) && \text{(Constant local memory lemma)} \\ &\qquad = \text{fst}([\![T]\!]_{\pi'}(l, m')) && \text{(Strong induction)} \\ &\qquad = \text{fst}([\![w_i \cdot T]\!]_{\pi'}(l, m')) && \text{(Constant local memory lemma)} \end{aligned}$$

  Similarly, for all $j \in [n]$, we perform a case split:

– $l_i \notin \mathbb{O}_\perp$:

$$\begin{aligned}
&\mathrm{snd}([\![w_i \cdot T']\!]_\pi(l,m))[j] \\
&= \mathrm{snd}([\![T]\!]_\pi(l, m[i \leftarrow \pi_{w_i}(l_i, m_i)]))[j] && \text{(definition)} \\
&= \mathrm{snd}([\![T]\!]_\pi(l, m[i \leftarrow \pi_{w_i}(l_i, \perp)]))[j] && \text{(Global Memory Irrelevance)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l, m'[i \leftarrow l_i]))[j], \perp) && \text{(strong induction)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l, m'[i \leftarrow \pi'_{w_i}(l_i, \perp)]))[j], \perp) && \text{(definition of } \pi') \\
&= \pi_{w_i}(\mathrm{snd}([\![w_i \cdot T]\!]_{\pi'}(l, m'))[j]) && \text{(definition of } \pi')
\end{aligned}$$

– $l_i \in \mathbb{O}_\perp$:

$$\begin{aligned}
\mathrm{snd}([\![w_i \cdot T']\!]_\pi(l,m))[j] &= \mathrm{snd}([\![T]\!]_\pi(l, m[i \leftarrow \pi_{w_i}(l_i, m_i)]))[j] && \text{(definition)} \\
&= \mathrm{snd}([\![T]\!]_\pi(l, m[i \leftarrow m_i]))[j] && \text{(Write After Commit)} \\
&= \mathrm{snd}([\![T]\!]_\pi(l, m))[j] \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l, m'))[j], \perp) && \text{(strong induction)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l, m'[i \leftarrow \pi_{w_i}(l_i, m'_i)]))[j], \perp) && \text{(Write After Commit)} \\
&= \pi_{w_i}(\mathrm{snd}([\![w_i \cdot T]\!]_{\pi'}(l, m'))[j]) && \text{(definition of } \pi')
\end{aligned}$$

- $T = r_i \cdot T'$: We have

$$\begin{aligned}
\mathrm{fst}([\![r_i \cdot T']\!]_\pi(l,m)) &= \mathrm{fst}([\![T]\!]_\pi(l[i \leftarrow \pi_{r_i}(l_i, m)], m)) && \text{(definition)} \\
&= \mathrm{fst}([\![T]\!]_{\pi'}(l[i \leftarrow \pi_{r_i}(l_i, m)], m')) && \text{(strong induction)} \\
&= \mathrm{fst}([\![T]\!]_{\pi'}(l[i \leftarrow \pi_{r_i}(l_i, (\pi_{w_0}(m'_0), ..., \pi_{w_{n-1}}(m'_{n-1})))], m')) && \text{(definition of } m') \\
&= \mathrm{fst}([\![T]\!]_{\pi'}(l[i \leftarrow \pi'_{r_i}(l_i, m')], m')) && \text{(definition of } \pi') \\
&= \mathrm{fst}([\![r_i \cdot T]\!]_{\pi'}(l, m')) && \text{(definition of } \pi')
\end{aligned}$$

Similarly, for all $j \in [n]$

$$\begin{aligned}
&\mathrm{snd}([\![r_i \cdot T']\!]_\pi(l,m))[j] \\
&= \mathrm{snd}([\![T]\!]_\pi(l[i \leftarrow \pi_{r_i}(l_i, m)], m))[j] && \text{(definition)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l[i \leftarrow \pi_{r_i}(l_i, m)], m'))[j]) && \text{(strong induction)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l[i \leftarrow \pi_{r_i}(l_i, (\pi_{w_0}(m'_0, \perp), ..., \pi_{w_{n-1}}(m'_{n-1}, \perp)))], m'))[j]) && \text{(definition)} \\
&= \pi_{w_i}(\mathrm{snd}([\![T]\!]_{\pi'}(l[i \leftarrow \pi'_{r_i}(l_i, m')], m'))[j]) && \text{(definition)} \\
&= \pi_{w_i}(\mathrm{snd}([\![r_i \cdot T]\!]_{\pi'}(l, m'))[j]) && \text{(definition of } \pi')
\end{aligned}$$

- $T = d_i \cdot T'$: We have

$$\begin{aligned}
\mathrm{fst}([\![d_i \cdot T']\!]_\pi(l,m)) &= \mathrm{fst}([\![\mathrm{proj}_{\neg i}(T')]\!]_\pi(l,m)) && \text{(definition)} \\
&= \mathrm{fst}([\![\mathrm{proj}_{\neg i}(T')]\!]_{\pi'}(l,m)) && \text{(strong induction)} \\
&= \mathrm{fst}([\![d_i \cdot T']\!]_{\pi'}(l,m)) && \text{(definition)}
\end{aligned}$$

Similarly, for all $j \in [n]$

$$\begin{aligned}
\mathrm{snd}([\![d_i \cdot T']\!]_\pi(l,m))[j] &= \mathrm{snd}([\![\mathrm{proj}_{\neg i}(T)]\!]_\pi(l,m))[j] && \text{(definition)} \\
&= \pi_{w_i}(\mathrm{snd}([\![\mathrm{proj}_{\neg i}(T')]\!]_{\pi'}(l,m'))[j], \perp) && \text{(strong induction)} \\
&= \pi_{w_i}(\mathrm{snd}([\![d_i \cdot T']\!]_{\pi'}(l,m'))[j], \perp) && \text{(definition)}
\end{aligned}$$

Now note that $\bot^n[i] = \bot = \pi_{w_i}(\bot^n[i], \bot)$. Consider arbitrary $l \in \mathbb{I}^n_\bot$ and trace $T \in \Gamma$. By assumption, there exists finite prefix T' of T satisfying $P(l, \text{fst}(\llbracket T' \rrbracket_\pi(l, \bot^n))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$. By lemma B.0.2, we now also have $P(l, \text{fst}(\llbracket T' \rrbracket_{\pi'}(l, \bot^n))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$. Thus concluding our proof that the full-disclosure protocol π' P-solves Θ under the non-layered model for Γ traces.

($\Leftarrow$) This direction is trivial because any full-disclosure protocol is also a protocol. □

Appendix C

Full Proof of the Non-layered Γ_{Nrs} to Non-layered Γ_{Valid} Proposition

Proposition C.0.1 (Non-layered Γ_{Nrs} to non-layered Γ_{Valid} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Nrs} traces, then Θ is also P-solvable under the non-layered model for Γ_{Valid} traces.

Proof. Suppose Θ is P-solvable under the non-layered model for Γ_{Nrs} traces. By the Full-disclosure proposition (see subsection 3.5.1), there exists a full-disclosure protocol $(\mathbb{V}, \pi)$ that P-solves Θ under the non-layered model for Γ_{Nrs} traces.

We construct a protocol $(\mathbb{V}, \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \pi_{w_i}(x, y)$$
$$\pi'_{r_i}(x, m) = \begin{cases} x & \text{if } x \in \mathbb{O}_\perp \text{ or } m_i = \perp \\ \pi_{r_i}(x, m) & \text{otherwise} \end{cases}$$

We claim that $(\mathbb{V}, \pi')$ P-solves Θ under the non-layered model for Γ_{Valid} traces.

Consider arbitrary trace $T \in \Gamma_{\text{Valid}}$. We can always split T into $T_1 \cdot T_2$ for some finite trace T_1 and trace T_2 where for all $i \in [n]$, $w_i \in T_1$ or $d_i \in T_2$. This is because T must contain at least one w_i or d_i for each i.

Consider the following rewriting system with a family of rules $(R_i)_i$ for each $i \in [n]$:

$$R_i : T_a r_i T_b \Rightarrow T_a T_b \qquad \text{where } T_a \text{ does not contain } w_i, d_i, \text{ or } r_i$$

Lemma C.0.2. The above rewriting system is terminating on all finite traces.

This is obvious since the single rule decreases the length of a trace by one. Since the length of all finite traces is finite, we can only apply the rule finitely many times.

Lemma C.0.3. For all $i \in [n]$, the rewriting system only containing R_i is confluent on all finite traces.

This is obvious since we can perform the rewrite on at most one place in the trace. If there are multiple r_i actions, we can only remove the first r_i (given no w_i and d_i actions are before it).

Let f be the function which takes in a finite trace, and returns the trace after repeatedly applying R_i until it is irreducible in order of i.

Lemma C.0.4. $\text{dead}(f(T_1) \cdot T_2) = \text{dead}(T_1 \cdot T_2)$

This is obvious because we do not remove nor add d_i for all $i \in [n]$ in the rewriting system.

Lemma C.0.5. $f(T_1) \cdot T_2 \in \Gamma_{\text{Nrs}}$

We prove this by considering $\text{proj}_i(f(T_1) \cdot T_2)$ for each $i \in [n]$. Note that only R_i might affect the projection, and not any other R_j where $i \neq j$. Notice first that R_i repeatedly removes the first r_i if it is the first action in the projection. After termination, it must be the case that the first action must not be a r_i. Moreover the rule R_i preserves validity of the trace.

Lemma C.0.6. For all $l \in \mathbb{V}^n$, $[\![f(T_1)]\!]_{\pi'}(l, \bot^n) = [\![T_1]\!]_{\pi'}(l, \bot^n)$

We prove this by induction on the number of rewriting steps applied.

- Base case: If no steps are applied, $f(T_1) = T_1$, and the equation is trivially satisfied.
- Inductive step: It suffices to prove that if $T_1 = T_a r_i T_b$ where T_a does not contain w_i, d_i, or r_i, $[\![T_a r_i T_b]\!]_{\pi'} = [\![T_a T_b]\!]_{\pi'}$.

  We have

  $$
  \begin{aligned}
  &[\![T_a r_i T_b]\!]_{\pi'}(l, \bot^n) \\
  &= ([\![r_i T_b]\!]_{\pi'} \circ [\![T_a]\!]_{\pi'})(l, m) && \text{(Compositionality of valid traces)} \\
  &= [\![r_i T_b]\!]_{\pi'}(\text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n)), \text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n))) \\
  &= [\![T_b]\!]_{\pi'}(\text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n)) \\
  &\quad [i \leftarrow \pi'_{r_i}(\text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n))[i], \text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n)))], \\
  &\quad \text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n)))
  \end{aligned}
  $$

  Notice that from the Constant global memory lemma for non-layered models, we have $\text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n))[i] = \bot^n[i] = \bot$.

  Thus we have

  $$
  \begin{aligned}
  &[\![T_a r_i T_b]\!]_{\pi'}(l, \bot^n) \\
  &= [\![T_b]\!]_{\pi'}(\text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n)) \\
  &\quad [i \leftarrow \text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n))[i]], \\
  &\quad \text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n))) && \text{(definition of } \pi') \\
  &= [\![T_b]\!]_{\pi'}(\text{fst}([\![T_a]\!]_{\pi'}(l, \bot^n)), \text{snd}([\![T_a]\!]_{\pi'}(l, \bot^n))) \\
  &= ([\![T_b]\!]_{\pi'} \circ [\![T_a]\!]_{\pi'})(l, m) \\
  &= [\![T_a T_b]\!]_{\pi'}(l, \bot^n) && \text{(Compositionality of valid traces)}
  \end{aligned}
  $$

Lemma C.0.7. For all finite valid traces $T, l \in \mathbb{V}^n, i \in [n]$, where $w_i \in T$, if $\text{snd}([\![T]\!]_{\pi'}(l, \bot^n))[i] = \bot$, then $\text{fst}([\![T]\!]_{\pi'}(l, \bot^n))[i] = \bot$.

This can be shown easily by induction on length of T.

Lemma C.0.8. For all finite prefixes T' of $(f(T_1)T_2)$, $l \in \mathbb{I}^n_\bot$, we have $[\![T']\!]_\pi(l, \bot^n) = [\![T']\!]_{\pi'}(l, \bot^n)$

We prove this by induction on length of T':

- $T' = \epsilon$**:** This is trivial.

- $T' = T'' \cdot w_i$**:** This follows trivially from the Compositionality of valid traces and the fact that $\pi'_{w_i} = \pi_{w_i}$.
- $T' = T'' \cdot r_i$**:** Recall that $f(T_1) \circ T_2 \in \Gamma_{\text{Nrs}}$ from lemma C. It follows that $w_i \in T''$. We have

$$
\begin{aligned}
&[\![T'' r_i]\!]_{\pi'}(l, \bot^n) \\
&= [\![r_i]\!]_{\pi'}([\![T'']\!]_{\pi'}(l, \bot^n)) && \text{(Compositionality of valid traces)} \\
&= [\![r_i]\!]_{\pi'}([\![T'']\!]_{\pi}(l, \bot^n)) && \text{(induction)} \\
&= (\text{fst}([\![T'']\!]_{\pi}(l, \bot^n)) \\
&\quad [i \leftarrow \pi'_{r_i}(\text{fst}([\![T'']\!]_{\pi}(l, \bot^n))[i], \text{snd}([\![T'']\!]_{\pi}(l, \bot^n))), \\
&\quad \text{snd}([\![T'']\!]_{\pi}(l, \bot^n)))
\end{aligned}
$$

  Here we perform a case split. If $\text{snd}([\![T'']\!]_{\pi}(l, \bot^n))[i] = \bot$, by lemma C.0.7, we have $\text{fst}([\![T'']\!]_{\pi}(l, \bot^n))[i] = \bot$, which is in $\mathbb{O}_{\bot}$. Thus we have:

$$
\begin{aligned}
&(\text{fst}([\![T'']\!]_{\pi}(l, \bot^n)) \\
&\quad [i \leftarrow \pi'_{r_i}(\text{fst}([\![T'']\!]_{\pi}(l, \bot^n))[i], \text{snd}([\![T'']\!]_{\pi}(l, \bot^n))), \\
&\quad \text{snd}([\![T'']\!]_{\pi}(l, \bot^n))) \\
&= (\text{fst}([\![T'']\!]_{\pi}(l, \bot^n)) \\
&\quad [i \leftarrow \pi_{r_i}(\text{fst}([\![T'']\!]_{\pi}(l, \bot^n))[i], \text{snd}([\![T'']\!]_{\pi}(l, \bot^n))), \\
&\quad \text{snd}([\![T'']\!]_{\pi}(l, \bot^n))) && \text{(semantics of protocol)} \\
&= [\![T'' r_i]\!]_{\pi}(l, \bot^n)
\end{aligned}
$$

  Otherwise, the equation is satisfied obviously.

- $T' = T'' \cdot d_i$**:** This is trivial.

Now, by definition of P-solvability, there exists a finite prefix T_p of $f(T_1)T_2$ satisfying $P(l, \text{fst}([\![T_p]\!]_{\pi}(l, \bot^n))[\text{dead}(f(T_1)T_2) \leftarrow \bot], \text{dead}(f(T_1)T_2), \Theta)$. By the Trace extension lemma, there exists a finite prefix T'_p of $f(T_1)T_2$ satisfying

$$P(l, \text{fst}([\![T'_p]\!]_{\pi}(l, \bot^n))[\text{dead}(f(T_1)T_2) \leftarrow \bot], \text{dead}(f(T_1)T_2), \Theta)$$

where $f(T_1)T_r = T'_p$ for some trace T_r.

We have:

$$
\begin{aligned}
&[\![T'_p]\!]_{\pi}(l, \bot^n) \\
&= [\![f(T_1)T_r]\!]_{\pi}(l, \bot^n) \\
&= [\![f(T_1)T_r]\!]_{\pi'}(l, \bot^n) && \text{(lemma C.0.8)} \\
&= ([\![T_r]\!]_{\pi'} \circ [\![f(T_1)]\!]_{\pi'})(l, \bot^n) && \text{(Compositionality of valid traces)} \\
&= ([\![T_r]\!]_{\pi'} \circ [\![T_1]\!]_{\pi'})(l, \bot^n) && \text{(lemma C.0.6)} \\
&= [\![T_1 T_r]\!]_{\pi'}(l, \bot^n) && \text{(Compositionality of valid traces}
\end{aligned}
$$

We observe that $T_1 T_r$ is a prefix of T, and together with lemma C.0.4, we have $P(l, \text{fst}([\![T_1 T_r]\!]_{\pi'}(l, \bot^n))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$, thus completing our proof.

□

Appendix D

Full Proof of the Non-layered $\Gamma_{\mathbf{Alt}}$ to Non-layered $\Gamma_{\mathbf{Nrs}}$ Proposition

Proposition D.0.1 (Non-layered Γ_{Alt} to non-layered Γ_{Nrs} proposition)**.** Given task description Θ and solvability property P, if Θ is P-solvable under the non-layered model for Γ_{Alt}, then Θ is also P-solvable under the non-layered model for Γ_{Nrs} traces.

Proof. Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ_{Alt} traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V} \cup \mathbb{N}$.

We define the function **ver**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{ver}(x) = \begin{cases} z & \text{if } x = \langle y, z \rangle \\ 0 & \text{otherwise} \end{cases}$$

We also define the function **val**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} y & \text{if } x = \langle y, z \rangle \\ x & \text{otherwise} \end{cases}$$

We define the protocol $(\mathbb{V}', \pi')$ as follows:

$$\pi'_{w_i}(x, y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\perp \\ \langle \pi_{w_i}(\mathbf{val}(x), \perp), \mathbf{ver}(x) \rangle & \text{otherwise} \end{cases}$$

$$\pi'_{r_i}(x, m) = \begin{cases} x & \text{if } x \in \mathbb{O}_\perp \\ x & \text{if } \mathbf{ver}(m[i]) \neq \mathbf{ver}(x) \\ \begin{cases} \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)) & \text{if } \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)) \in \mathbb{O}_\perp \\ \langle \pi_{r_i}(\mathbf{val}(x), \mathbf{map}(\mathbf{val}, m)), & \\ \mathbf{val}(x) + 1 \rangle & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

We continue our proof in two steps. We first prove that $(\mathbb{V}', \pi')$ P-solves Θ under the non-layered model for Γ_{Alt} traces. We then prove that it also P-solves Θ under the non-layered model for Γ_{Nrs} traces as required.

For our first claim, we define the following invariant $I \subseteq \mathbb{V}^n \times \mathbb{V}^n \times \mathbb{V}'^n \times \mathbb{V}'^n \times \mathbb{A}^*$. We define that $I(l, m, l', m', T)$ holds iff:

- The two global memories agree with respect to value, i.e. for all $i \in [n]$, $m[i] = \mathbf{val}(m'[i])$
- For all $i \in [n]$, one of these cases hold:
  1. $l_i = l'_i$ and $l_i \in \mathbb{O}_\perp \cup \mathbb{I}$
  2. $d_i \in T$
  3. w_i is the last action in $\text{proj}_i(T)$, $l_i = \mathbf{val}(l'_i)$, $\mathbf{ver}(m'_i) = \mathbf{ver}(l'_i)$, and $l_i \notin \mathbb{O}_\perp$
  4. r_i is the the last action in $\text{proj}_i(T)$, $l_i = \mathbf{val}(l'_i)$, $\mathbf{ver}(m'_i) + 1 = \mathbf{ver}(l'_i)$, and $l_i \notin \mathbb{O}_\perp$

We now prove this lemma:

Lemma D.0.2. For all traces $T \in \Gamma_{\text{Alt}}$, prefixes T' of T, $l_{\text{initial}} \in \mathbb{I}^n_\perp$, let $(l, m) = [\![T']\!]_\pi(l_{\text{initial}}, \perp^n)$ and $(l', m') = [\![T']\!]_{\pi'}(l_{\text{initial}}, \perp^n)$, we have $I(l, m, l', m', T')$.

We prove this by induction on T'. Note that T and subsequently T' are valid traces, so we make use of the Compositionality of valid traces implicitly:

- $T' = \epsilon$**:** It is trivial to see that the two global memories agree with each other. For each $i \in [n]$, case 1 holds.
- $T' = T''d_i$**:** It is trivial to see the two global memories still agree by the induction hypothesis and the Constant global memory lemma for non-layered models. For i, case 2 holds trivially. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4).
- $T' = T''w_i$**:** We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider are cases 1 and 4, since $T \in \Gamma_{\text{Alt}}$.

  If case 1 is true before the write action, by the Write After Commit condition and the Constant local memory lemma, both the local and global memories remain unchanged and thus the two global memories agree, and case 1 holds.

  If case 4 is true, unfolding the definitions and considering the Constant global memory lemma for non-layered models, the two global memories agree and case 3 holds.
- $T' = T''r_i$**:** We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider for the T'' prefix are cases 1 and 3 since $T \in \Gamma_{\text{Alt}}$.

  If case 1 is true before the read action, by the Read After Commit condition and the Constant global memory lemma for non-layered models, both the local and global memories remain unchanged and thus the two global memories agree and case 1 holds.

  If case 3 is true before the read action, by the Constant global memory lemma for non-layered models, the two global memories remain unchanged and agree with each other. We also perform a case split. If $\pi_{r_i}(l_i, m) \in \mathbb{O}_\perp$, by unfolding the definitions, case 1 holds. Otherwise case 4 holds.

Lemma D.0.3. The protocol $(\mathbb{V}', \pi')$ P-solves Θ under the non-layered model for Γ_{Alt} traces.

Consider an arbitrary trace $T \in \Gamma_{\text{Alt}}$ and $l \in \mathbb{I}^n_\perp$. By assumption, there exists a prefix T' of T such that $P(l, \text{fst}([\![T']\!]_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$. By considering lemma D.0.2, we then have $P(l, \text{fst}([\![T']\!]_{\pi'}(l, \perp^n))[\text{dead}(T) \leftarrow \perp], \text{dead}(T), \Theta)$ as required.

Consider the following rewriting system with a family of rules $R_{a,i}$ for $a \in \{r, w\}$ and $i \in [n]$ defined as follows:

$$R_{w,i} : T_a w_i T_b w_i T_c \Rightarrow T_a w_i T_b T_b \qquad \text{where } T_b \text{ does not contain any actions of } i$$
$$R_{r,i} : T_a r_i T_b r_i T_c \Rightarrow T_a r_i T_b T_b \qquad \text{where } T_b \text{ does not contain any actions of } i$$

Lemma D.0.4. The rewriting system is terminating on all finite traces.

This is obvious since each rule decreases the length of a trace by one. Since the length of a finite trace is finite, we can apply rules finitely many times.

Lemma D.0.5. The above rule is confluent on all finite traces.

This result is a consequence by analyzing all critical pairs of the rewriting system and observing that they all are convergent.

Let f be the function that takes in a finite trace T and returns the trace after being rewritten by the above rewriting system.

Lemma D.0.6. For all finite traces T_1, T_2 with $T_1 \leq T_2$, we have $f(T_1) \leq f(T_2)$.

This can be proven by induction on the length of the traces.

Lemma D.0.7. For all finite valid traces T, $[\![T]\!]_{\pi'} = [\![f(T)]\!]_{\pi'}$.

This can be proven by induction on the number of rules applied. The Compositionality of valid traces and constant memory lemmas (see subsection 2.2.4) are used in the proof.

We now define f' which takes in a trace $T \in \Gamma_{\text{Nrs}}$ and returns $\bigsqcup_i f(T_i)$ for some increasing sequence of finite traces $(T_i)_i$ such that $\bigsqcup_i T_i = T$.

Lemma D.0.8. For all traces T, $\text{dead}(f'(T)) = \text{dead}(T)$

This is obvious because we never add nor remove d_i actions in the rewriting rule.

Lemma D.0.9. For all traces $T \in \Gamma_{\text{Nrs}}$, $f'(T) \in \Gamma_{\text{Alt}}$.

This can be proved by proving that each $f'(T_i)$ is alternating by induction and that the limit of a sequence of alternating traces must also be alternating.

Consider arbitrary $l \in \mathbb{I}^n_\perp$ and trace $T \in \Gamma_{\text{Nrs}}$. We have $f'(T) \in \Gamma_{\text{Alt}}$ by lemma D.0.9. By definition of P-solvability, there exists a prefix T' of $f'(T)$ such that $P(l, \text{fst}([\![T']\!]_{\pi'}(l, \perp^n))[\text{dead}(f(T)) \leftarrow \perp], \text{dead}(f(T)), \Theta)$.

We have

$$\begin{aligned} & T' \\ \leq\ & f'(T) && \text{(definition)} \\ =\ & \bigsqcup_i f(T_i) && \text{(definition)} \end{aligned}$$

for some increasing sequence of finite traces $(T_i)_i$, where $T = \bigsqcup_i T_i$. Thus, there must exists some $k \in \mathbb{N}$ such that $T' \leq f(T_k)$.

By the Trace extension lemma, we have $P(l, \mathrm{fst}(\llbracket f(T_k) \rrbracket_{\pi'}(l, \bot^n))[\mathrm{dead}(f'(T)) \leftarrow \bot], \mathrm{dead}(f'(T)), \Theta)$. By simple substitution, we also have $P(l, \mathrm{fst}(\llbracket T_k \rrbracket_{\pi'}(l, \bot^n))[\mathrm{dead}(T) \leftarrow \bot], \mathrm{dead}(T), \Theta)$, thus completing the proof. □

Appendix E

Full Proof of the δ-protocol Proposition

Proposition E.0.1 (δ-protocol proposition)**.** For any distributed task Θ, trace property $\Gamma \subseteq \Gamma_{k\text{-Alt}}$ for some $k \in \mathbb{N}$, and solvability property P, Θ is P-solvable under the non-layered model for Γ traces if and only if there exists a δ-protocol ϕ that P-solves Θ under the non-layered model for Γ traces. This proposition is also true in the layered case.

Proof. We only prove this for the non-layered case. The layered case is similar.

($\Rightarrow$) : Assume the protocol $(\mathbb{V}, \pi)$ P-solves Θ under the non-layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V}$.

We define the functions **val**, **his**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} a & \text{if } x = \langle a, b \rangle \\ x & \text{otherwise} \end{cases}$$
$$\mathbf{his}(x) = \begin{cases} b & \text{if } x = \langle a, b \rangle \\ \bot & \text{otherwise} \end{cases}$$

We define the δ-protocol $(\mathbb{V}', \phi)$ as follows:

$$\phi_{w_i}(x) = \pi_{w_i}(\mathbf{val}(x), \mathbf{his}(x))$$
$$\phi_{r_i}(x, m) = \begin{cases} \bot & \text{if } x = \bot \\ \langle \pi_{r_i}(\mathbf{val}(x), m), m \rangle & \text{otherwise} \end{cases}$$
$$\phi_{\delta_i}(x) = \mathbf{val}(x)$$

We now define the following invariant $I \subseteq \mathbb{V}^n \times \mathbb{V}^n \times \mathbb{V}'^n \times \mathbb{V}'^n \times \mathbb{A}^*$ where $I(l, m, l', m', T)$ holds iff:

- The two global memories are identical, i.e. $m = m'$
- The local memories agree with each other, i.e. $l[i] = \mathbf{val}(l'[i])$
- For all $i \in [n]$, one of these cases hold:
    1. $l'[i] = \bot$ and $m'[i] = \bot$

2. The last action of $\text{proj}_i(T)$ is a read or does not exist, $l'[i] \neq \bot$ and $\mathbf{his}(l'[i]) = m'[i]$
3. The last action of $\text{proj}_i(T)$ exists and is not a read, and $l'[i] \neq \bot$

We now prove this lemma:

Lemma E.0.2. For all traces $T \in \Gamma$, prefixes T' of T, $l_{\text{initial}} \in \mathbb{I}_\bot^n$, let $(l, m) = [\![T']\!]_\pi(l_{\text{initial}}, \bot^n)$ and $(l', m') = [\![T']\!]_\phi(l_{\text{initial}}, \bot^n)$, we have $I(l, m, l', m', T')$.

We prove this by induction on the length of T'. Note that T and subsequently T' are valid traces, so we make use of the Compositionality of valid traces implicitly:

- $T' = \epsilon$: It is trivial to see that two first two conditions are satisfied. Depending whether $l_{\text{initial}}[i] = \bot$, we have the first case or the second for each $i \in [n]$.
- $T' = T''d_i$: By the constant memory lemmas (see subsection 2.2.4) and the induction hypothesis, the conditions are satisfied.
- $T' = T''w_i$: We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider are cases 1 and 2, since $T \in \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the write action, by the Write After Commit condition and the Constant local memory lemma, case 1 holds.

  If case 2 holds before the write, by unfolding the definitions, we have case 3 afterwards.
- $T' = T''r_i$: We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider are cases 1 and 3, since $T \in \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the read action, by the Read After Commit condition and the Constant global memory lemma for non-layered models, case 1 holds.

  If case 3 holds before the read, by unfolding the definitions, we have case 2 afterwards.

Now consider arbitrary trace $T \in \Gamma$ and $l \in \mathbb{I}_\bot^n$. By assumption, there exists a prefix T' of T such that $P(l, \text{fst}([\![T']\!]_\pi(l, \bot^n))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$. By the Trace extension lemma, we have $P(l, \text{fst}([\![T]\!]_\pi(l, \bot^n))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$. By lemma E.0.2, we have:

$$P(l, \phi_\delta(\text{fst}([\![T]\!]_\phi(l, \bot^n)))[\text{dead}(T) \leftarrow \bot], \text{dead}(T), \Theta)$$

($\Leftarrow$): Assume the δ-protocol $(\mathbb{V}, \phi)$ P-solves Θ under the non-layered model for Γ traces. We define $\mathbb{V}'$ to be the set freely generated by the binary operator $\langle \_, \_ \rangle$ on $\mathbb{V} \cup \mathbb{N}$.

We define the functions **val**, **ver**: $\mathbb{V}' \to \mathbb{V}'$ as follows:

$$\mathbf{val}(x) = \begin{cases} a & \text{if } x = \langle a, b \rangle \\ x & \text{otherwise} \end{cases}$$
$$\mathbf{ver}(x) = \begin{cases} b & \text{if } x = \langle a, b \rangle \\ 0 & \text{otherwise} \end{cases}$$

We define the normal protocol $(\mathbb{V}', \pi)$ as follows:

$$\pi_{w_i}(x, y) = \begin{cases} y & \text{if } x \in \mathbb{O}_\perp \\ \phi_{w_i}(\mathbf{val}(x)) & \text{otherwise} \end{cases}$$

$$\pi_{r_i}(x, m) = \begin{cases} x & & \text{if } x \in \mathbb{O}_\perp \\ \begin{cases} \phi_{\delta_i}(\phi_{r_i}(\mathbf{val}(x), m)) & \mathbf{ver}(x)\text{=k-1} \\ \langle \phi_{r_i}(\mathbf{val}(x), m), \mathbf{ver}(x) + 1 \rangle & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

We now define the following invariant $I \subseteq \mathbb{V}^n \times \mathbb{V}^n \times \mathbb{V}'^n \times \mathbb{V}'^n \times \mathbb{A}^*$ where $I(l, m, l', m', T)$ holds if and only if:

- The two global memories are identical, i.e. $m = m'$
- For $i \in [n]$, one of these cases hold:
  1. $l[i] = l'[i] = m[i] = \perp$
  2. $l[i] = l'[i] \in \mathbb{I}$ and $\text{proj}_i(T)$ contains no reads from i
  3. $l'[i] = \langle l[i], q \rangle$ where q is the number of reads in $\text{proj}_i(T)$ and $q \neq k$
  4. $l'[i] = \phi_{\delta_i}(l[i])$ and the number of reads in $\text{proj}_i(T)$ is k

We now prove this lemma:

Lemma E.0.3. For all traces $T \in \Gamma$, prefixes T' of T, $l_{\text{initial}} \in \mathbb{I}^n_\perp$, let $(l, m) = [\![T']\!]_\phi(l_{\text{initial}}, \perp^n)$ and $(l', m') = [\![T']\!]_\pi(l_{\text{initial}}, \perp^n)$, we have $I(l, m, l', m', T')$.

We prove this by induction on the length of T'. Note that T and subsequently T' are valid traces, so we make use of the Compositionality of valid traces implicitly:

- $T' = \epsilon$: It is trivial to see that two global memories agree. Depending whether $l_{\text{initial}}[i] = \perp$, we have the first case or the second for each $i \in [n]$.
- $T' = T''d_i$: By the constant memory lemmas (see subsection 2.2.4) and the induction hypothesis, the conditions are satisfied.
- $T' = T''w_i$: We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider are cases 1, 2, and 3, since $T \in \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the write action, by the Write strictness condition and the Constant local memory lemma, case 1 holds.

  If case 2 holds before the write, by unfolding the definitions, we have case 2 afterwards.

  If case 3 holds before the write, by unfolding the definitions, we have case 3 afterwards.
- $T' = T''r_i$: We first assume the induction hypothesis for the T'' prefix. For all $j \in [n] \setminus \{i\}$, the same case holds by the constant memory lemmas (see subsection 2.2.4). For i, the cases we need to consider are cases 1, 2, and 3, since $T \in \Gamma_{k\text{-Alt}}$.

  If case 1 is true before the write action, by the Read strictness condition and the Constant global memory lemma for non-layered models, case 1 holds.

  If case 2 or 3 is true before the read action, we perform a case split on whether $\mathbf{ver}(x) = k - 1$. By unfolding the definitions, we have either case 4 or case 3 holding.

Now consider arbitrary trace $T \in \Gamma$ and $l \in \mathbb{I}_{\perp}^{n}$.

By assumption, we have $P(l, \phi_{\delta}(\mathrm{fst}([\![T]\!]_{\phi}(l, \perp^{n})))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$. By lemma E.0.3, we have $P(l, \mathrm{fst}([\![T]\!]_{\pi}(l, \perp^{n}))[\mathrm{dead}(T) \leftarrow \perp], \mathrm{dead}(T), \Theta)$ as required. □

Appendix F

GMT's Well-bracketed Property

Consider the following definition from GMT's paper:

Definition F.0.1 (Well-bracketed traces)**.** A well-bracketed trace T is an alternating trace where for all $i \in [n]$, if $\text{proj}_i(T)$ contains a d_i action, (which must be the last action since T is valid), the second last action must be a w_i.

The trace property $\Gamma_{\text{Wb+It+IS}}$ contains all well-bracketed traces $T \in \Gamma_{\text{It+IS}}$.

We show that we can solve a distributed task with the layered model for $\Gamma_{\text{Wb+It+IS}}$ traces if and only if we can solve the same task with the layered model for $\Gamma_{\text{It+IS}}$ traces.

Proposition F.0.1 (Equivalence of layered $\Gamma_{\text{Wb+It+IS}}$ and layered $\Gamma_{\text{It+IS}}$ proposition)**.** Given task description Θ, Θ is P_{GMT}-solvable under the layered model for $\Gamma_{\text{Wb+It+IS}}$ traces if and only if it is P_{GMT}-solvable under the layered model for $\Gamma_{\text{It+IS}}$ traces.

Proof. We prove both directions as follows:

($\Rightarrow$): Assume the protocol $(\mathbb{V}, \pi)$ P_{GMT}-solves Θ under the layered model for $\Gamma_{\text{Wb+It+IS}}$ traces. We claim that $(\mathbb{V}, \pi)$ also P_{GMT}-solves Θ under the layered model for $\Gamma_{\text{It+IS}}$ traces.

Consider arbitrary trace $T \in \Gamma_{\text{It+IS}}$ and $l \in \mathbb{I}^n_\perp$. Note that there must exists traces T_1, T_2 such that $T_1 T_2 = T$ and $d_i \in T_1$ for all $i \in \text{dead}(T)$.

Consider a family of rewriting systems R_i indexed by $i \in [n]$ with two rules each:

$$R1_i : T_a r_i T_b d_i T_c \Rightarrow T_a d_i T_b T_c \qquad \text{where } T_b \text{ does not contain } w_i, d_i, \text{ or } r_i$$
$$R2_i : T_a d_i T_b \Rightarrow w_i d_i T_a T_b \qquad \text{where } T_a \text{ does not contain } w_i, d_i, \text{ or } r_i$$

Lemma F.0.2. For each $i \in [n]$, the rewriting system R_i is both terminating and confluent for all finite alternating traces.

This is obvious since exactly one of the two rules can be applied iff the last action of the i-th process is d_i, and the second last action is not w_i. After the rewrite, this condition does not hold.

Now let f_i be the function that takes in a finite alternating trace and perform the rewriting system R_i on it.

Lemma F.0.3. For all prefixes T_1 of T, there exists a set $s \subseteq \text{dead}(T)$ such that $\text{fst}(\llbracket T_1 \rrbracket_\pi(l, \perp^n))[\text{dead}(T) \leftarrow \perp] = \text{fst}(\llbracket (f_{n-1} \circ \cdots \circ f_0)(T_1) \rrbracket_\pi(l[s \leftarrow \perp], \perp^n))[\text{dead}(T) \leftarrow \perp]$.

We prove this by induction on the number of f_i's applied.

- **Base case:** We take $s = \emptyset$ and the equation holds trivially.
- **Inductive case:** Let $T_1' = (f_{k-1} \circ \cdots \circ f_0)(T_1)$. By the induction hypothesis, suppose $\mathrm{fst}(\llbracket T_1 \rrbracket_\pi(l, \bot^n))[\mathrm{dead}(T) \leftarrow \bot] = \mathrm{fst}(\llbracket T_1' \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T) \leftarrow \bot]$.

  We perform a case split:

  - $R1_k$ **is applied on** T_1'**:** It suffices to prove that $\mathrm{fst}(\llbracket T_a r_i T_b d_i T_c \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T) \leftarrow \bot][i] = \mathrm{fst}(\llbracket T_a d_i T_b T_c \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T) \leftarrow \bot][i]$ for all $i \in [n]$. We perform another case split:
    - $i \in \mathbf{dead}(T)$: This is trivial as LHS = $\bot$ = RHS.
    - $i \notin \mathbf{dead}(T)$: It suffices to prove that $\mathrm{fst}(\llbracket T_a r_i T_b d_i T_c \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[i] = \mathrm{fst}(\llbracket T_a d_i T_b T_c \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[i]$ which we can prove by induction easily on the length of $T_b T_c$ and showing that the global memories are identical and that the local memories except the i-th one agree.
  - $R2_k$ **is applied on** T_1'**:** It suffices to prove that $\mathrm{fst}(\llbracket T_a d_i T_b \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T) \leftarrow \bot][i] = \mathrm{fst}(\llbracket w_i d_i T_a T_b \rrbracket_\pi(l[s \cup \{i\} \leftarrow \bot], \bot^n))[\mathrm{dead}(T) \leftarrow \bot][i]$ for all $i \in [n]$.

    This is similar to the previous case, where we perform a case split on whether $i \in \mathrm{dead}(T)$. In the first case it is trivial. In the second case, one just have to prove the global memory and the local memory except the i-th element agree by induction on the length of T_a and T_b.
  - **No rules are applied on** T_1'**:** This is trivial as $f_k(T_1') = T_1'$.

Let $T_1' = (f_{n-1} \circ \ldots \circ f_0)(T_1)$.

Lemma F.0.4. $T_1' T_2 \in \Gamma_{\mathrm{Wb+It+IS}}$.

It suffices to show that $T_1' T_2$ is well bracketed, iterated, immediate snapshot. We implicitly make use of the relatively obvious fact that $\mathrm{proj}_i(T_1' T_2) = \mathrm{proj}_i(f_i(T_1) T_2)$.

It is obvious that $T_1' T_2$ is well-bracketed by the definition of R_i. One can also observe that it is iterated by considering the numbering of the actions and that each rewriting rule preserves the iterated property. Lastly, by case analysis of each rule, one can observe that each rewriting rule preserves immediate snapshot.

It now suffices to claim that there exists prefix T_P' of T_2 such that we have

$$(l[\mathrm{dead}(T_1 T_2) \leftarrow \bot], \mathrm{fst}(\llbracket T_1 T_p' \rrbracket_\pi(l, \bot^n))[\mathrm{dead}(T_1 T_2) \leftarrow \bot]) \in \Theta$$

We define the finite set S as the powerset of $\mathrm{dead}(T_1)$. Note that by assumption, for all $s \in S$, there exists a prefix T_p of $T_1' T_2$ such that $(l[s \leftarrow \bot][\mathrm{dead}(T_1' T_2) \leftarrow \bot], \mathrm{fst}(\llbracket T_p \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(f(T_1) T_2) \leftarrow \bot]) \in \Theta$. By the Trace extension lemma and the fact that S is finite, there must exists prefix T_P' of T_2 such that for all $s \in S$, we have $(l[s \leftarrow \bot][\mathrm{dead}(T_1' T_2) \leftarrow \bot], \mathrm{fst}(\llbracket T_1' T_p' \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T_1' T_2) \leftarrow \bot]) \in \Theta$.

By considering the rewriting rules, it is obvious that $\mathrm{dead}(T_1' T_2) = \mathrm{dead}(T_1') = \mathrm{dead}(T_1) = \mathrm{dead}(T_1 T_2)$. Therefore for all $s \in S$, we have $(l[\mathrm{dead}(T_1 T_2) \leftarrow \bot], \mathrm{fst}(\llbracket T_1' T_p' \rrbracket_\pi(l[s \leftarrow \bot], \bot^n))[\mathrm{dead}(T_1 T_2) \leftarrow \bot]) \in \Theta$. We note that $T_1 T_p'$ is a prefix of T, and that $(f_{n-1} \circ \cdots \circ f_0)(T_1 T_p') = (f_{n-1} \circ \ldots \circ f_0)(T_1) T_p' = T_1' T_p'$. So finally by lemma F.0.3, we have $(l[\mathrm{dead}(T_1 T_2) \leftarrow \bot], \mathrm{fst}(\llbracket T_1 T_p' \rrbracket_\pi(l, \bot^n))[\mathrm{dead}(T_1 T_2) \leftarrow \bot]) \in \Theta$ as required.

($\Leftarrow$)**:** This direction follows trivially from the Trace subset proposition.

□