UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN CIENCIAS MATEMÁTICAS Y DE LA ESPECIALIZACIÓN EN ESTADÍSTICA APLICADA

ON SOME INTERACTIONS BETWEEN TOPOLOGY AND COMPUTABILITY

TESIS
QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN CIENCIAS

PRESENTA:
RODOLFO CONDE MARTÍNEZ

TUTOR PRINCIPAL:
DR. MARCELO ALBERTO AGUILAR GONZÁLEZ DE LA VEGA
INSTITUTO DE MATEMÁTICAS, UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

COMITÉ TUTOR:
DR. SERGIO RAJSBAUM GORODEZKY   DR. FRANCISCO GONZÁLEZ ACUÑA
INSTITUTO DE MATEMÁTICAS, UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

CIUDAD DE MÉXICO, FEBRERO 2016

On some interactions between topology and computability

Rodolfo Conde Martínez
rodolfo@matem.unam.mx

Febrero 2016

Para el Señor mi Dios, Voluntad, Verbo y Fuerza. Porque Él siempre me ha bendecido y ahora su bendición para su siervo es enorme a través de la familia que me ha dado, nos ayudó en los momentos mas difíciles y me ha guiado de nuevo a través del camino de la ciencia, abriendo mi entendimiento para permitirme sentir la hermosa sensación del descubrimiento matemático.

A mi hermosha esposa Yanira, por su inigualable amor e interminables paciencia y apoyo durante este camino, ella me ha demostrado la fortaleza y la determinación de una mujer y de una madre en la lucha por el bienestar de sus hijos y su familia. Ella me ha hecho el hombre mas feliz del universo al convertirme en padre y dar nacimiento a los dos hermosos frutos de nuestro amor.

A mi amada hija Andrea Eunice*, valiente y hermosa pequeña mía, por tu sonrisa y luz que me han dado fuerzas para no abandonar el camino.*

A mi amado hijo Rodolfo Gabriel*, por la enorme enseñanza que Dios nos ha dado con tu llegada. Por ti hemos aprendido a luchar con valor y tener esperanza.*

A mi padre Rodolfo Conde del Águila, que siempre me brinda su apoyo incondicional y espera con gran ansia la culminación de mi preparación.

Hoy y siempre...GRACIAS.

Conoceréis la verdad; y la verdad os hará libres.

Juan 8.32

El principio de la sabiduría es el temor a Dios;
Los insensatos desprecian la sabiduría y la enseñanza.

Proverbios 1.7

En Dios siempre hay esperanza.

Agradecimientos

A Dios mi Señor, siempre primero, por guiarme en mi camino y una vez mas sumergirme en el enorme mar del conocimiento matemático. Y no solo eso, pues durante la realización de mi doctorado, Dios me ha dado mi propia familia al casarme con mi esposa, me ha convertido en padre de unos hermosos hijos y finalmente, me ha permitido conocer la sensación del descubrimiento, de la investigación, con todos sus altibajos. Ojala pueda seguir recorriendo este camino toda mi vida.

A mi pequeño hijo Rodolfo Gabriel (Júnior !!!), él es una gran sorpresa que llega justo en la etapa final de mi doctorado, y hemos aprendido tanto con su llegada. Él también se ha convertido en uno de mis motivos principales para terminar este trabajo y convertirme en un mejor hombre. Mamá, sus hermanos y yo lo esperamos con mucha alegria, ya queremos que esté jugando con nosotros y pueda finalmente sentir la sensación de la libertad. Se que con la ayuda de Dios, él va a nacer con bien y total salud, y mamá también va a estar bien, y los dos llegarán juntos a su hogar y un día, Rodolfo Gabriel podrá leer estas lineas que le dedico. Te amo hijo mío.

A mi pequeña hija Andrea Eunice (mejor conocida como Palomina), por que Dios la trajo a nuestra familia con bien, con salud y una gran alegría, en medio del proceso de escritura de esta tesis. Ella fue y sigue siendo uno de mis motivos principales para concluir este trabajo y seguir adelante con todos mis sueños y anhelos, no solamente para poder tener la capacidad de dar sustento económico a toda nuestra familia, sino también para ser un hombre fuerte, completo y feliz, y así poder transmitirle a ella lo mejor de mi, y ayudar a que ella también sea feliz. Te amo hija mía.

A mi hijo Ariyan, por recibirme con los brazos abiertos y ser el primero en apoyar la unión de su mami y un servidor. Ariyan ha sido un gran apoyo en esta etapa de la historia de nuestra familia, mientras cuidamos a mamá y nos preparamos para la llegada de su camarada. Espero realmente poder ayudarle para que alcance todas las bendiciones que Dios tiene preparadas para el.

A mi amada esposa, Yanira Gózalez Ramos, por todo su inmenso amor, paciencia y cuidados desde que nos conocimos hasta el día de hoy. Si había alguien interesado en que no tirara la toalla en mi investigación, es precisamente ella. Con la llegada de nuestros hijos y todo lo que ha resistido por ellos, me ha mostrado el significado real de dar la vida por los hijos y sin duda sólo una madre como ella es capaz de hacer lo que ella ha hecho con Andrea Eunice y lo que ahora hace por Rodolfo Gabriel. Te amo chiquita mosha.

A mis padres. Rodolfo Conde del Águila, mi papá que me guió no solamente con sus palabras y consejos, sino también con su ejemplo y su determinación y coraje en su trabajo y en cumplir todos sus deberes como padre con su familia. María de los Ángeles Martínez, mi mamá que cuando estuve cerca de ella me cuidaba, mantenía la casa limpia para todos y siempre esta preocupada por sus hijos.

A Chendy, por el cariño y apoyo que siempre me brinda, y por ser una excelente esposa para mi papa, una súper madre para mis hermanos Perla y Carlos y una gran madrina para mi hija Andrea Eunice.

A toda mi familia, mis hermanas y hermano Thalía, Perla, Viridiana y Carlos, por muchos momentos de diversión. Mis abuelos, tíos y primos. En particular, todo mi amor y agradecimiento a mi abuelita Aurea del Águila† y mi tío José Guadalupe Conde del Águila, el Tío Pipe†. También quiero dedicar estas últimas lineas de este agradecimiento a la memoria de mi primo José Antonio Martínez Miranda†. Que Dios te tenga en su gloria, Toño.

A la familia de mi esposa, por la hospitalidad inigualable que me mostraron siempre que iba a visitarlos. Mis suegros, que siempre son muy amables conmigo y nos aconsejan a Yani y a mi de como llegar a los cuarenta años de casados. Nos ayudaron mucho al inicio de nuestro matrimonio, apoyandonos con Andrea Eunice y ahora, en esta etapa de nuestra vida, mientras cuidamos a Yani en la espera de nuestro hijo Rodolfo Gabriel, mis suegros han sido un apoyo invaluable en muchos aspectos. Muchas gracias a mi suegra la Sra América, por la gran ayuda que nos brindo ese día tan dificíl, sin duda Dios nos ayudo a través de ella, nunca lo olvidaré. Mi cuñada Karina (Margara !!) y su esposo Carlos, Ixchel, Joselito y por supuesto, mi querido cuñado Edgar (Bebole) y su familia, gracias por su confianza, apoyo y unas pláticas muy interesantes.

A todos mis amig@s, que desde la primaria hasta hoy día cada un@ han compartido una parte de mi vida. Mis hermanos Miguel, Marcos, Paco y Galphord siempre tendrán toda mi amistad y profundo agradecimiento.

A mis tutores, por todo su esfuerzo, paciencia y apoyo para ayudarme a preparar mi mente y mi conocimiento para desarrollar mis temas de investigación. Ellos son, en orden de aparición:

- Marcelo Aguilar, mi tutor principal, siempre me ha guiado en mi camino a través de la topología y la teoría de la computación. Bastaba con que Marcelo me escuchara solo unos minutos acerca de lo que estaba intentando hacer para que él de inmediato supiera que articulo o libro debía de consultar para poder avanzar en mi trabajo sobre variedades y computación. He aprendido mucho de Marcelo desde la maestría hasta el día de hoy, realmente espero se me haya pegado al menos un poquito de su enorme capacidad de abordar la matemática y en especial la topología.

- Sergio Rajsbaum, parte de mi comité tutoral, quien con mucha paciencia y esmero me mostró la emocionante (pero muy intrincada) rama de las ciencias de la computación llamada Computación Distribuida y su bella conección con la topología. Con el tema que desarrolle con la ayuda de Sergio, viví mi primer experiencia en una conferencia internacional (GETCO 2010), exponiendo temas de sistemas distribuidos, y como resultado final de esta experiencia, salio el primer artículo relacionado con mi doctorado, y después de este, trabajamos muy duro para poder poner a la luz nuestro trabajo sobre la "cota inferior". Con los consejos y paciencia de Sergio, ese trabajo alcanzo un nivel de entendimiento excepcional. También le agradezco la oportunidad de participar en la segunda reunión conjunta SMM-RSME en 2012.

- Francisco González Acuña, parte de mi comité tutoral. Ha sido para mi un gran honor que el profesor Fico haya aceptado ser mi sinodal en mi examen profesional de maestría y luego ser uno de mis cotutores durante mi doctorado. Todas las veces que tuve la oportunidad de platicar con el profesor Fico y escuchar sus enseñanzas y recomendaciones aprendí mucho sobre la teoría general de variedades, sus relaciones (respecto a computación) con grupos y los muchos problemas resueltos y no resueltos sobre decidibilidad en variedades. De hecho, nuestro trabajo sobre variedades computables comenzó buscando algo que el profesor Fico sugirió tratar de formular: "Presentaciones finitas para variedades topológicas".

A mis sinodales, por el valioso tiempo que dedicaron a mi y a la evaluación de mi trabajo, gracias por todas sus críticas, correcciones y recomendaciones sobre los temas de investigación de mi tesis.

A cada réferi anónimo que le dedico tiempo y esfuerzo para conocer mi trabajo en los artículos que escribí con la ayuda de mis tutores sobre variedades, computación y sistemas distribuidos.

Sus correcciones y sugerencias fueron un ingrediente muy valioso para enriquecer el contenido de los artículos y esta tesis.

A los miembros del grupo de Complejidad y Computabilidad en Análisis (CCA, por sus siglas en ingles, con dirección web http://cca-net.de/list.html), por su gran ayuda, comentarios, correcciones y sugerencias sobre análisis y topología computable, todo esto fue un ingrediente muy importante para completar el trabajo de variedades computables. En particular, gracias a Vasco Brattka, Klaus Weihrauch, Mathieu Hoyrup y Dimiter Skordev.

A mi Alma Máter, la Universidad Nacional Autónoma de México, el Instituto de matemáticas y nuestra querida Facultad de ciencias. En estos lugares comenzó esta historia académica con una carrera en computación, una maestría en ciencias matemáticas y el día de hoy se finaliza la etapa del proyecto de investigación que realice con la ayuda de mis tutores. Deseo profundamente en mi corazón que esta historia no termine aquí en el doctorado, y que continue el resto de mi vida.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo económico que me otorgo por cuatro años, sin el cual hubiera sido prácticamente imposible poder dedicarme a entender y desarrollar el proyecto de doctorado que culmina en esta tesis.

Al proyecto PAPIIT-UNAM IN108702 por el apoyo económico que me proporciono durante un año para llevar a cabo la finalización de mi proyecto de doctorado.

A Martín Gutierrez y Samuel Gutierrez, por creer en mi y por su enorme apoyo al brindarme un trabajo seguro, estable y también me dieron un gran apoyo durante la finalización de los preparativos de la celebración de mi boda.

A Henry Kemper y Walter Kemper, por brindarme la oportunidad de trabajar proyectos de programación muy interesantes y por la paciencia que tuvieron para mi forma de trabajar.

A todas las personas en el mundo entero que colaboran en los sitios web de Wikipedia, la enciclopedia libre (http://www.wikipedia.org), el sitio de preguntas y respuestas de matemáticas de StackExchange (http://math.stackexchange.com), Mathworld (http://mathworld.wolfram.com) y PlanetMath (http://planetmath.org). Todos son excelentes sitios web, donde siempre encontré información y referencias iniciales de varias cosas cubiertas en esta obra. Son muy buenos puntos de inicio para buscar conceptos que no conocemos. Los recomiendo ampliamente, e invito a todos a cooperar con estos sitios, para que sigan mejorando cada día.

Para todos ustedes, que tienen una huella indeleble en mi vida, no me queda mas que decirles

GRACIAS

Abstract for Part I

In the first part of this thesis, we build a basic foundation to develop a theory of computability on topological manifolds. We propose a definition for the abstract term “computable manifold” (with boundary) by introducing computability as a structure that we impose to a given topological manifold, just in the same way as differentiability or piecewise linearity are defined for smooth and PL manifolds respectively. Using the framework of computable topology and Type-2 theory of effectivity, we develop computable versions of all the basic concepts needed to define manifolds, like *computable atlases* and *(computably) compatible* computable atlases. We prove that given a computable atlas Φ defined on a set M, we can construct a computable topological space $(M, \tau_\Phi, \beta_\Phi, \nu_\Phi)$, where τ_Φ is the topology on M induced by Φ and that the equivalence class of this computable space characterizes the *computable structure* determined by Φ. The concept of *computable submanifold* is also investigated. We show that any compact computable manifold with boundary which satisfies a computable version of the T_2-separation axiom, can be embedded as a computable submanifold of some euclidean space $\mathbb{R}^q$, with a computable embedding, where $\mathbb{R}^q$ is equipped with its usual topology and some canonical computable encoding of all open rational balls.

Abstract for Part II

The theory of distributed computing shares a deep and fascinating connection with combinatorial and algebraic topology. One of the key ideas that facilitates the development of the topological theory of distributed computing is the use of *iterated* shared memory models. In such a model processes communicate through a sequence of shared objects. Processes access the sequence of objects, one-by-one, in the same order and asynchronously. Each process accesses each shared object only once. In the most basic form of an iterated model, any number of processes can crash, and the shared objects are snapshot objects. A process can write a value to such an object, and gets back a snapshot of its contents. It is known that the consensus task and the (n,k)-set agreement task cannot be implemented in the iterated model.

In the second part of this thesis, we first introduce the basic *iterated shared memory model of distributed computing* and explain how its runs can be described in a topological manner with simplicial complexes. Then we define and describe in detail, what we call an extension of the iterated model based on the *safe-consensus* task (Afek, Gafni and Lieber, DISC'09). In a safe-consensus task, the validity condition of consensus is weakened as follows. If the first process to invoke an object solving a safe-consensus task returns before any other process invokes it, then the process gets back its own input; otherwise the value returned by the task can be arbitrary. As with consensus, the agreement requirement is that always the same value is returned to all processes. In the new iterated model with safe-consensus, the processes repeatedly: write their information to a (fresh) shared memory array, invoke safe-consensus boxes and snapshot the contents of the shared array. We show with examples that the topology of simplicial complexes representing executions of protocols in the iterated model with safe-consensus is very different from the case of the basic iterated model. Also, we investigate the solvability of the (n,k)-set agreement task and the consensus task in the iterated model with safe-consensus. Our main results are (1) The consensus task can be implemented in the iterated model with safe-consensus, using $\binom{n}{2}$ safe-consensus invocations; (2) The $\binom{n}{2}$ upper bound is tight: Any implementation of consensus in the iterated model with safe-consensus tasks needs at least $\binom{n}{2}$ safe-consensus invocations. The lower bound proof uses an intricate bivalency argument and also some combinatorial arguments based on connectivity of subgraphs of *Johnson graphs*.

Introduction

In this thesis, we develop new connections between two mathematical fields: the theory of computation and topology. Our results can be divided into two main parts, the first part deals specifically with the development of a theory of computability on topological manifolds; while in the second part we apply results from topology and combinatorics to obtain new results in the theory of distributed systems. All the results presented in the second part of this thesis were published for the first time in [CR12, CR14].

Part I: Computable manifolds

Computability theory over continuous structures began formally in 1936 with the landmark paper of Alan Turing [Tur36] where he defined the notion of a single computable real number: $x \in \mathbb{R}$ is computable if its decimal expansion can be calculated in the discrete sense, that is, output by a Turing machine. Since then, other authors have developed definitions and results to try to build a reasonable theory of computability in the continuous setting. There are two main approaches to modeling computations with real number inputs. The first approach is given by the framework of *Computable Analysis* studied in many papers and some books, e.g., [Lac55, Grz57, PER89, Ko91, Wei00]. The second approach is the algebraic one, its development goes back to the 1950s and focuses on the algebraic operations needed to perform tasks [BM75, BCS97]. The most influential model is the so called *BSS model* developed by Blum, Shub and Smale [BSS89, BCSS98].

Computable Analysis reflects the fact that computers can only store finite amounts of information. Since real numbers and other objects in analysis are "infinite" in nature, a Turing machine can only use finite objects to approximate them and perform the actual computations on these finite pieces of information, thus we have that topology plays an important role in computable analysis [Wei00]. The representation approach and the framework of *Type-2 Theory of Effectivity* (TTE) [KW85], a generalization of ordinary computability theory, has provided a solid background to formalize the theory of computable analysis [Wei00]. TTE has been extensively studied and developed as a standalone topic in computability theory [Wei08]. Also, it has been generalized to a more general model of computability for analysis [TW11]. *Computable metric spaces* (also known as *recursive metric spaces*) [Wei93, Bra03] have been defined in computable analysis and play a most important role in the subject, as many results of computability over euclidean spaces can be generalized to the broader world of computable metric spaces. For an overview of basic computable analysis, see the tutorial given in [BHW08].

In recent years, as a product of various publications [Wei00, Sch03, GW05, GSW07] trying to consider computable topology as a foundation of computable analysis, Weihrauch and Grubba [WG09] developed a solid foundation for computability over more general spaces, where the main objects of study are called *computable topological spaces*. Roughly speaking, a computable topological space is a T_0-space (X, τ) in which a base $\beta \subseteq \tau$ is provided with an encoding with strings from Σ^*, such that the set of all valid strings that encode elements of β is computable and under this encoding, intersection of base elements is computable (in a formal sense, see Definition 2.1.1). This framework has been used to prove many important results in computable topology, like the following:

- Dini's Theorem states that if X is a compact topological space and $\{f_n\}$ is a monotonically increasing sequence (i.e., $f_n(x) \leqslant f_{n+1}(x)$ for all n and x) of continuous functions on X which converges pointwise to a continuous function f, then the convergence is uniform. In the paper [GW05], a computable version of Dini's Theorem is proven.

- A celebrated result in general topology and the theory of metric spaces states that every second-countable regular topological space (X, τ) is metrizable, that is, X is homeomorphic to a metric space (M, d). In [GSW07], it is proven that every computable topological space satisfying a "computably regular" condition, has a computable embedding in a computable metric space (which topologically is its completion). Also, a computable Urysohn Lemma is proven.

- A computable version of the Stone-Weierstrass approximation Theorem is constructed in [GWX08] for computable topological space which satisfy computable versions of the Hausdorff and locally compact properties.

The paper [WG09] is an effort to put in one place the most general and basic facts about computable topology, because these facts were scattered throughout many papers (e.g., [Wei00, GW05, GSW07, GWX08] among others). Computable versions of topological properties and separation axioms are an important tool to obtain computable versions of important topological theorems and the paper [Wei10] gives definitions of many computable versions of each separation axiom and proves the relations between these computable separation axioms. More recent results of computable topology include [RW13], where Rettinger and Weihrauch study computability aspects of finite and infinite products of computable topological spaces and they prove computable versions of Tychonoff's Theorem.

With the advent of computable topological spaces and the solid foundation of computability in euclidean spaces provided by computable analysis, the stage is set to introduce computability properties in a very popular class of topological spaces: *Topological manifolds*. A space M is a topological manifold (with boundary) if and only if each point $x \in M$ has a neighborhood homeomorphic to an open set of euclidean half space $\mathbb{R}^n_+$ for fixed n (this integer is called the *dimension* of M). Manifolds are one of the most important types of topological spaces, many problems related to manifolds have been inspiration for some of the most beautiful mathematical constructions, using very sophisticated and advanced techniques, as it can be seen in the foundational work of Kirby and Siebenmann [KS77] and more recently, in the paper [Man13], where Manolescu shows that there exist high-dimensional topological manifolds which cannot be triangulated as simplicial complexes, thus refuting the *Triangulation conjecture* [GS80]. But despite the fact that there is a lot of knowledge about topological manifolds, there are still unanswered questions and hard open problems related to them. As a result of this, manifold theory is always intensively studied by the mathematicians doing research in topology.

Additional structure can be imposed to a topological manifold, giving rise to special classes of manifolds, like *smooth manifolds* [Whi36, BC70], *analytic manifolds* [Hör90] and *piecewise linear (PL) manifolds* [Hud69, RS82]. The relationships between manifolds equipped with any or all of these structures and standard topological manifolds are revised in many papers [Whi36, Ker60, Sma61, KS77, Fre82].

Our contributions. In this thesis, computability enters the world of topological manifolds, we propose a definition of the abstract term "computable manifold" by introducing computability as a structure that we impose to a given topological manifold, just in the same way as differentiability or piecewise linearity are defined for smooth and PL manifolds respectively. Using the framework of computable topology and TTE, we develop effective versions of the constructs needed to define manifolds e.g., *charts* and *atlases*. Namely, the following concepts are worked out:

- The definition of *computable atlas* Φ on a set M;
- the computable topological space induced by Φ on M;
- the definition of *computably compatible computable atlases*;
- the definition of *computable structure*, which is only a equivalence class of computably compatible computable atlases (characterized by the equivalence class of a computable topological space, see Definition 2.2.12).

A *computable manifold* is just a set M endowed with a computable structure. We present many examples of computable manifolds and their respectively computable topological spaces. We study the relationships of computable manifolds and computable functions (with respect to the representations induced by computable atlases) and prove that computable homeomorphisms and computable structures behave nicely when working together. We also prove some simple (but very useful) basic properties of computable manifolds.

Submanifolds, that is, manifolds which are inside other manifolds are an important tool to investigate manifolds. We define *computable submanifolds* and present some properties about them. Finally, we use all the previous definitions and results about computable manifolds to give an effective version of the following well known result concerning topological manifolds [Gau74, Mun00]: *Every compact Hausdorff topological manifold embeds in some high dimensional euclidean space*. We show that any compact computable manifold M which is also computably Hausdorff (see Definition 2.3.1) can be embedded in some euclidean space $\mathbb{R}^q$, where q depends on M and we equip $\mathbb{R}^q$ with its usual topology and some standard computable encoding of all open rational balls. Thus every compact computable manifold that is computably Hausdorff can be seen as a computable submanifold of some euclidean space.

Related work. In [CM09], Calvert and Miller give another definition for the term "computable manifold". Using the BSS model [BCSS98], $\mathbb{R}$-*computable manifolds* are defined in [CM09]. Informally, a $\mathbb{R}$-computable manifold is a topological manifold M, together with a finite collection of $\mathbb{R}$-computable functions (e.g. computable in the BSS sense), called the inclusion functions, which describe the inclusion relations of all the domains of the charts of a specific atlas $\{\varphi_i\colon U_i \to \mathbb{R}^d\}_{i\in\mathbb{N}}$ on M. This definition is used to prove some results (in the BSS model) about the undecidability of nullhomotopy and simple connectedness in $\mathbb{R}$-computable manifolds and also how to determine a presentation of the fundamental group of such manifolds. However, it is hard to interpret these undecidability results in terms of practical computing, because in the BSS model, simple subsets of $\mathbb{R}^2$ which can be easily "drawn", such as the Koch snowflake and the graph of $y = e^x$ are undecidable[1] in the BSS model[BCSS98], thus it would seem that in general, uncomputability results in the BSS model do not match real world computations.

In [Ilj13], Iljazović studies the computability (in the sense of TTE and computable analysis) of compact subsets of computable metric spaces. They use these results to show that each semi-computable compact manifold with computable boundary is computable, as a subset of a metric space. In [BI14], the authors prove similar results for 1-manifolds, not necessary compact. In this thesis, our focus is to develop the basic concepts and results necessary to build an effective theory of manifolds.

[1]If a set $C \subseteq \mathbb{R}^n$ is decidable in the BSS model, then it must be a countable disjoint union of semi-algebraic sets. This is the reason why simple sets such as the graph of $y = e^x$ are not decidable in the BSS model.

Part II: Distributed systems

A *distributed system* consists of *n processes* (computers, small sensors, network nodes, etc.) which communicate using some sort of communication device (shared memory, message passing, ...) that satisfies properties about timing and failure. These processes execute a *protocol* (i.e., an algorithm) to try to solve a specific *task*. The theory of distributed computing is an actively developed field of computer science that shares a deep and fascinating connection with combinatorial and algebraic topology. One of the key ideas that facilitates the development of the topological theory of distributed computing is the use of *iterated* shared memory models, introduced in [BG97]. In such a model, processes communicate through a sequence of shared objects. Processes access the sequence of objects, one-by-one, in the same order and asynchronously. Each process accesses each shared object only once.

In the most basic form of an iterated model, any number of processes can crash and the shared objects are snapshot objects [AAD$^+$93]. A process can write a value to such an object, and gets back a snapshot of its contents. It is known that this model is equivalent to the standard wait-free read/write shared memory model [BG97, GR10a], but its runs are more structured and easier to analyze than the runs in the standard shared memory model. The recursive nature of the iterated shared memory model was instrumental for the results in [BG97] and for the proof of the *Asynchronous computability theorem* of [HS99]. This theorem uncovered the intimate connection that exists between topology and distributed computing. Extensions of the basic iterated model have also been studied, where the processes communicate through a sequence of objects more powerful than snapshot objects [GRH06] or where the asynchrony of the system is limited to model failure detectors [RRT08b]. For an overview of the iterated approach, the reader can refer to [Raj10].

Our purpose in this thesis is to enrich the theory of iterated models. Using as a basis the current knowledge about iterated models and the connections with combinatorial topology, we define a new extension of the iterated model, by adding to the basic model the ability to invoke shared objects more powerful than the snapshot objects. Specifically, the new shared objects are based on the properties of the *safe-consensus* task of [AGL09], a recently proposed variation of the consensus task.

In the *consensus* task, each process receives as input a private value such that all correct processes eventually output a value (Termination); all the processes output the same value (Agreement); and the output value must be a value proposed by some process (Validity). The consensus task is of fundamental importance in distributed computing, it is one of the most studied distributed task. In the paper [Her91] Herlihy examined the power of different synchronization primitives for *wait-free* computation, e.g., when computation completes in a finite number of steps by a process, regardless of how fast or slow other processes run, and even if some of them halt permanently. He showed that consensus is a universal primitive, in the sense that a solution to consensus (with read/write registers) can be used to implement any synchronization primitive in a wait-free manner. Also, consensus cannot be wait-free implemented from read/write registers alone [FLP85, LAA87]; indeed, all modern shared-memory multiprocessors provide some form of universal primitive.

The impossibility of solving consensus using a shared memory composed of read/write registers is one of the most important discoveries in distributed systems. This result holds even in systems where only one process may crash [FLP85, LAA87]. Given the huge practical importance of consensus, many situations have been studied to ameliorate the implications of this impossibility result, from stronger models of distributed computing, to weaker versions of consensus. A popular example of this is the (n, k)-set agreement task [Cha93], a weakening of consensus. In this task, n processes receive private input values and they output values such that Termination and Validity of consensus are satisfied, but the Agreement property is replaced by k-Agreement: The set of output values of all correct processes must be of size no bigger than k. It was proven in STOC'93 by three independent teams [BG93a, HS93, SZ93] and using topological methods, than (n, k)-set agreement cannot be solved in the iterated model with snapshot objects. This result holds even when $k = n - 1$. Because the iterated model of distributed computing is equivalent to the standard wait-free shared memory model [BG97, GR10a], this result is also true in the standard model. The impossibility of set agreement in shared memory systems was a surprising discovery, because it is also known that the set agreement task is strictly weaker that the consensus task.

Afek, Gafni and Lieber introduced the *safe-consensus* task in [AGL09], which seemed to be a synchronization primitive much weaker than consensus. In the safe-consensus task, the processes agree on an unique output value that must satisfy the Termination and Agreement properties of consensus, but the validity requirement becomes the following: if the first process to invoke the task returns before any other process invokes it, then it outputs its input; otherwise, when there is concurrency, the consensus output can be arbitrary, not even the input of any process. In any case, all processes must agree on the same output value. Trivially, consensus can implement safe-consensus. Surprisingly, Afek et all [AGL09] showed that the converse is also true, by presenting a wait-free implementation of consensus using safe-consensus black-boxes and read/write registers. Thus, consensus and safe-consensus are wait-free equivalent, because one can use one task as a black box to implement the other. Why is it then, that safe-consensus seems a much weaker synchronization primitive?

Our contributions. We use the iterated model of [BG97] and the safe-consensus task of [AGL09] to define a new model of distributed computing and we investigate some of the topological properties of this new model and the solvability of the consensus and set agreement tasks. The iterated model defined in this thesis, is an extension of the iterated model of [BG97] with the power of *safe-consensus objects*. A *safe-consensus object* is a shared object that receives input values from the processes and returns to the processes output values consistent with the safe-consensus task specification. The new model allows the processes to communicate by using a sequence of snapshot objects and safe-consensus objects. As with any iterated model, computation proceeds in rounds, accessing copies of the objects, asynchronously, and in the same order. In each iteration, each process first writes to the shared memory, then it invokes a safe-consensus object and finally it takes a snapshot of the shared memory.

First, we define in detail the new iterated model of distributed computing, enriched with the safe-consensus shared objects, we call it the *iterated model with safe-consensus* objects. Once we have introduced to the reader the new model, we begin to investigate some of its properties. In general, it is difficult to work with shared objects more powerful than shared memory, because even if our protocols are specified in an iterated form, the model can lose its recursive nature because of the invocations made to the shared objects, thus it can be very difficult to prove something useful about the new model, all because of the power added by the new shared objects. Thus, our strategy to begin our work with the iterated model with safe-consensus objects is the following. We will put a (temporally) additional restriction in the iterated model with safe-consensus:

RSC Each safe-consensus object in the sequence must be invoked by all the processes.

(In the general iterated model with safe-consensus, the processes can invoke different safe-consensus objects in each round of a protocol). We call this model of computation the *iterated shared memory with full safe-consensus objects* (IFSC) model. It has good advantages to begin our research of the iterated model with safe-consensus objects with the addition of the restriction **RSC**, it will allow us to introduce basic techniques in an easy way and also it will be easier to study the topological properties of the new model, so that we will be able to obtain a good deal of information on the new model, even with the restriction **RSC**. Indeed, we obtain our first contributions in the IFSC model, by analyzing the topology of this model and the solvability of the $(n, n-1)$-set agreement and the consensus tasks. Specifically, the first results we present are:

- The $(n, n-1)$-set agreement task can be implemented in the IFSC model using only one safe-consensus object (with a simple one-round protocol, Theorem 6.3.2);

- It is impossible to solve the consensus task for n processes in the IFSC model (Theorem 6.3.6).

These results say that the IFSC model is indeed more powerful than the basic iterated model, because $(n, n-1)$-set agreement cannot be solved using only shared memory [BG93a, HS93, SZ93]. But the IFSC model still has limitations as it cannot solve consensus (while consensus is solvable using safe-consensus objects [AGL09] without restrictions). We will see that this impossibility does not come from the fact that we are working in an iterated model, rather, it is caused by the restriction RSC of the IFSC model.

Also, in connection with the topological theory of distributed computing to represent executions of protocols in the iterated model (and in the extended IFSC model), we will have something to say about Theorem 6.3.6. It is known that the protocol complexes in the iterated shared memory model are connected and because of this, it is impossible to solve consensus [BMZ90, HS99]. In this thesis, we argue that the protocol complexes in the IFSC model are disconnected, yet, these protocols cannot solve consensus. In summary, our aims with the IFSC model, include explaining:

1. How to analyze protocol complex connectivity in an iterated model;

2. how does the power of the communication objects affect the connectivity of a protocol complex;

3. and what are the consequences of connectivity for task solvability.

After we have completed our work with the IFSC model, we are ready to remove the restriction **RSC** and turn back to the iterated model with safe-consensus objects without restrictions. In this model, we focus entirely our attention at the solvability of the consensus task. Our results with the iterated model with safe-consensus objects are the following:

- We present a simple protocol to solve the consensus task in the iterated model with safe-consensus objects, using $\binom{n}{2}$ safe-consensus shared objects. This protocol is based on the new *g-2coalitions-consensus* task, a new distributed task that we propose, which may be of independent interest.

- We prove that any wait-free implementation of consensus for n processes in the iterated model with safe-consensus requires $\binom{n}{2}$ safe-consensus black-boxes (Theorem 9.3.6 and Corollary 9.3.7).

Thus, we show that while consensus and safe-consensus are wait-free equivalent, safe-consensus is a weaker communication primitive than consensus, as long as complexity is concerned. These results also say that the impossibility to solve the consensus task in the IFSC model comes precisely because of restriction **RSC** and not from the fact that we are working in an iterated model.

Our main result is the lower bound on the number of safe-consensus objects needed to solve consensus. It uses connectivity arguments based on subgraphs of *Johnson graphs* [DL98], and an intricate combinatorial and bivalency argument, that yields a detailed bound on how many safe-consensus objects of each type (fan-in) are used by the implementation protocol. To our knowledge, this is the first time in which an attempt is made to study an iterated model by adding objects as powerful as the consensus task and also it is the first time that Johnson graphs are used in distributed computing.

Related work. Researchers in distributed computing theory have been concerned from early on with understanding the relative power of synchronization primitives. The wait-free context is the basis to study other failure models e.g. [BGLR01], and there is a characterization of the wait-free, read/write solvable tasks [HS99]. As we have said before, the weakening of consensus, *set agreement*, where n processes may agree on at most $n-1$ different input values, is still not wait-free solvable [BG93a, HS99, SZ00] with read/write registers only. The *renaming* task where n processes have to agree on at most $2n-1$ names has also been studied in detail e.g. [ABND$^+$90, CRR11, CnHR12, CnR12b].

Iterated models e.g. [BG97, HR10b, HR13, Raj10, RRT08a, RRT08b] are an important and useful tool in distributed systems, they facilitate impossibility results, and (although more restrictive) facilitate the analysis of protocols [GR10b]. We follow in this thesis the approach of [GRH06] that used an iterated model to prove the separation result that set agreement can implement renaming but not vice-versa, and expect our result can be extended to a general model using simulations , as was done in [GR10a] for the separation result of [GRH06]. For an overview of the use of topology to study computability, including the use of iterated models and simulations see [HKR13].

Afek, Gafni and Lieber [AGL09] presented a wait-free protocol that implements consensus using $\binom{n}{2}$ safe-consensus black-boxes (and read/write registers). Since our implementation uses the weaker, iterated form of shared-memory, it is easier to prove correct. The safe-consensus task was used in [AGL09] to show that the g-tight-group-renaming task [AGL$^+$08] is as powerful as consensus for g processes.

The idea of the classical consensus impossibility result [FLP85, LAA87] is (roughly speaking) that the executions of a protocol in such a system can be represented by a graph which is always connected. The connectivity invariance has been proved in many papers using the critical state argument introduced in [FLP85], or sometimes using a layered analysis as in [MR02]. Connectivity can be used also to prove time lower bounds e.g. [ADLS94, DM90, MR02]. We extend here the layered analysis to prove a lower bound on the number of objects needed to implement consensus.

Outline of the thesis

This thesis is organized as follows. Part I is devoted to the study of computable manifolds and is divided into four chapters. Chapter 1 gives a brief introduction to the basic definitions and results on topology and standard computability theory. Extended material on these topics can be found on any standard textbook about each subject [Arm83, KW85, Eng89, Sip96, Koz97, Mun00, Coo04]. It is in this chapter where we introduce to the reader the world of *Type-2 theory of effectivity* (TTE)

[Wei00, BHW08, Wei08], an extension of basic theory of computability. TTE will allow us to build programs which can manipulate objects represented by an infinite amount of information. In Chapter 2 we give a short overview of the field of *Computable topology*. The essential definition of *computable topological space* is constructed and many definitions and results related to this concept are presented. The framework of TTE is used to build ways to represent points, open, closed and compact subsets of a computable space. The concept of *computable function* between computable topological spaces is defined, together with other useful constructs, like computable subspaces, computable homeomorphisms and embeddings. Many technical results are proven in this chapter. In Chapter 3 we put at work all the definitions and results of previous chapters to build our definition of *computable manifold*. We do this by first introducing *computable charts* and *computable atlases* and with these concepts at hand, we can formulate the definition of *computable structure* (an equivalence class of computable atlases on a given set). We show that a computable structure on a set X is characterized by a computable topology on X (just in the same way a manifold structure on X is characterized by a corresponding topology on X). A computable manifold will be defined as a set M together with a computable structure. We also prove many basic results about computable manifolds, which are computable versions of well known results of standard manifold theory. In Chapter 4 we introduce *computable submanifolds* and we present the main result of the first part of this thesis, which is an embedding theorem for compact computable manifolds: Any compact n-dimensional computable manifold (satisfying a computable Hausdorff axiom) can be embedded in $\mathbb{R}^q$ with a computable embedding, where q is sufficiently large and depends on n. Our conclusions for the first part can be found in Section 10.1 of Chapter 10.

Part II is concerned with the iterated shared memory model of distributed computing extended with the power of safe-consensus tasks and contains five chapters. In Chapter 5, we begin our study of distributed system, first we introduce the most basic definitions needed to work with distributed systems, (i.e. processes, registers, protocols, events, executions, distributed tasks [BG93a, HS99, SZ00, BGLR01, AR02, Gaf09, Raj10]) and after that, we define the basic *iterated shared memory model of distributed computing* (called simply the *iterated model*). The remainder of the chapter deals with the properties of the iterated model. We show how the recursive nature of the protocols written in the iterated model allow us to interpret the executions of a protocol (evolving in time) as simplicial complexes (*protocol complexes*). At the end of the chapter, it is argued that the connectivity of complexes representing executions of a protocol, is a fundamental property to determine if a protocol in the iterated model (or any other model) will be able to solve a given task. The last result of the chapter is a short proof of the impossibility to solve the consensus task [FLP85, LAA87] with a protocol in the iterated model. In Chapter 6 we give extended versions of many definitions of Chapter 5, so that we are able to define a new model of distributed computing, which we call *the iterated model with safe-consensus shared objects*, this new model extends the operations of reading and writing to the shared memory of the system, with the ability to invoke shared objects that satisfy the properties of the safe-consensus task. We show how the topology of the protocol complex of a protocol in the iterated model with safe-consensus, is very different from the topology of the protocol complex of a protocol in the basic iterated model, being the connectivity of the complexes the big difference. We end this chapter with two interesting results: (1) We prove the existence of a correct protocol in the iterated model with safe-consensus objects, which can solve the $(n, n-1)$-set agreement task (in one round !!); (2) The last result is the impossibility to solve the consensus task in the iterated model with safe-consensus objects, if we restrict the use of the safe-consensus tasks in the extended iterated model in the following way: In each iteration of a protocol, all processes must invoke the same safe-consensus object. in Chapter 7, we show that in the general iterated model with safe-consensus (without the previous restriction), we can solve the consensus task with an iterated protocol that uses $\binom{n}{2}$ safe-consensus objects. Although the given protocol is a bit complicated in its formal specification, the intuitive ideas behind its execution are easy to understand and we explain them in detail. The correctness proof of the protocol is well structured, thanks to the recursive nature of the iterated model. It is in this chapter where we introduce the g-2coalitions-consensus task, an intermediate task that we use in the formal specification of the protocol that solves consensus. In Chapter 8, we begin our road to prove the main result of the second part of this thesis: A (matching) lower bound on the number of safe-consensus objects needed to solve the consensus task with a protocol in the iterated model with safe-consensus. This lower bound shows that although consensus and safe-consensus are wait-free equivalent, complexity-wise they are very different, being safe-consensus a weaker communication primitive than consensus. Our lower bound result will be obtained by counting in how many different ways the processes must be able to invoke safe-consensus shared objects in the executions of a protocol. Chapter 8 contains many technical results that describe the structure of the iterated protocols with safe-consensus objects. This results give us information about the 1-dimensional connectivity of the protocol complexes of iterated protocols with safe-consensus. Thus we do not use the framework of topology and simplicial complexes explicitly, instead we rely on simple combinatorics and graph theory. In Chapter 9 we obtain the lower bound result, by combining the results from Chapter 8 with some results regarding the connectivity of subgraphs of *Johnson graphs* [DL98]. Our conclusions for the second part can be found in Section 10.2 of Chapter 10.

Contents

Abstract for Part I ix

Abstract for Part II xi

Introduction xiii

Part I: Computable manifolds 1

1 Topology and computability theory 5

1.1 Topology . . . 5

1.1.1 Topological manifolds . . . 6

1.1.2 Charts and atlases . . . 7

1.1.3 Submanifolds and embeddings . . . 11

1.2 Computability theory . . . 11

1.2.1 Basic notions of computability theory . . . 11

1.2.2 Turing machines . . . 12

1.2.3 Tarsky's decision method for the elementary algebra of the reals . . . 14

1.3 Type-2 theory of effectivity . . . 15

1.3.1 Type-2 machines and computable string functions . . . 15

1.3.2 Topologies for Σ^* and Σ^ω . . . 19

1.3.3 Multi-representations and representations . . . 21

1.3.4 The smn and utm properties . . . 27

2 Computable topology 29

2.1 Computability inside topological spaces . . . 29

2.1.1 Computable T_0 spaces . . . 29

2.1.2 Predicate spaces . . . 34

2.2 Computable functions . . . 38

2.2.1 Multi-representations for continuous functions . . . 39

2.2.2 Equivalences between computable topological spaces . . . 42

2.2.3 Computable embeddings . . . 47

2.3 Computably Hausdorff spaces . . . 47

3 Computable manifolds 49

3.1 Computable structures . . . 49

3.1.1 The computable predicate space induced by a topological atlas . . . 49

3.1.2 Computable atlases . . . 51

3.2 Morphisms between computable manifolds . . . 61

3.2.1 Computable functions . . . 61

3.3 Properties of computable manifolds . . . 63

3.3.1 Topological and computable properties . . . 63

3.3.2 Some special computable atlases . . . 65

4 Computable embeddings of manifolds **69**
4.1 Submanifolds and embeddings . . . 69
4.1.1 Computable submanifolds . . . 69
4.1.2 Embeddings of computable manifolds . . . 72
4.2 Embedding computable manifolds in euclidean spaces . . . 72
4.2.1 A simple embedding theorem . . . 73

Part II: Distributed systems **77**

5 The iterated shared memory model **81**
5.1 Basic definitions . . . 81
5.1.1 Distributed systems . . . 81
5.1.2 Events and round schedules . . . 82
5.1.3 Protocols and executions . . . 83
5.1.4 Decision tasks . . . 84
5.2 Two popular models of distributed computing . . . 85
5.2.1 The standard model . . . 85
5.2.2 The iterated model . . . 86
5.3 The geometry of the Iterated model . . . 87
5.3.1 The case of 3-process protocols . . . 88
5.3.2 The protocol complex . . . 89
5.4 Consensus and set agreement tasks . . . 90
5.4.1 The set agreement task . . . 91
5.4.2 The impossibility of consensus in the iterated model . . . 91

6 An enrichment of the iterated model **97**
6.1 A new iterated model . . . 98
6.1.1 The general extended iterated model . . . 98
6.1.2 The extended iterated model with safe-consensus . . . 100
6.2 The Iterated model with Full Safe-consensus objects . . . 102
6.2.1 The IFSC model . . . 102
6.2.2 The protocol complex with safe-consensus . . . 103
6.3 Solving set agreement tasks in the IFSC model . . . 105
6.3.1 A protocol for $(n, n-1)$-set agreement . . . 105
6.3.2 Impossibility of consensus in the IFSC model . . . 107

7 Implementing consensus with safe-consensus **113**
7.1 Coalitions consensus tasks . . . 113
7.1.1 Definition of the g-2coalitions-consensus task . . . 113
7.1.2 Safe-consensus implements g-2coalitions-consensus . . . 114
7.2 An iterated consensus protocol . . . 116
7.2.1 Basic properties of the consensus protocol . . . 116
7.2.2 The protocol . . . 117
7.3 Correctness of the consensus protocol . . . 119
7.3.1 Invariants of the protocol . . . 119
7.3.2 2Coalitions-consensus implements consensus . . . 121

8 The structure of iterated protocols with safe-consensus 123
8.1 Preliminaries 123
8.1.1 Further terminology of the iterated model with safe-consensus 123
8.2 The Stairway method 125
8.2.1 The construction of the Stairway method 125
8.2.2 Local connectivity of iterated protocols with safe-consensus 134

9 A lower bound to implement consensus 139
9.1 The combinatorics of iterated protocols with safe-consensus 139
9.1.1 Subgraphs of Johnson graphs 140
9.1.2 Combinatorial properties of a consensus protocol 140
9.1.3 The connectivity of subgraphs of $J_{n,m}$ 143
9.2 The combinatorics of the lower bound 146
9.2.1 The degree of indistinguishability of a path and safe-consensus invocations . 146
9.3 The formal results 150
9.3.1 Results for $3 \leqslant m \leqslant n$ 150
9.3.2 Results for $m = 2$ 156
9.3.3 The lower bound proof 156

10 Conclusions 159
10.1 Computable manifolds 159
10.1.1 Further work and open problems 159
10.1.2 Structures on topological manifolds and computability 160
10.2 The iterated model extended with safe-consensus 160
10.2.1 The IFSC model, consensus and set agreement 160
10.2.2 The iterated model with safe-consensus and the lower bound 161

List of Figures

1.1 A Turing machine with three tapes. 13
1.2 A Type-2 machine computing $y_0 = f_{\mathfrak{M}}(y_1, \ldots, y_k)$. 16
1.3 The topological space $(\{0,1\}^\omega, \tau_C)$. 19
1.4 The subtrees representing the open set $001\Sigma^\omega \cup 011\Sigma^\omega \cup 10\Sigma^\omega \cup 11\Sigma^\omega$. 20
1.5 The function $h\colon \subseteq Y \to Y_0$ realizes the multi-function $f\colon \subseteq M \rightrightarrows M_0$. 24

5.1 General form of a protocol in the iterated model 86
5.2 The protocol complex for a 3-process protocol in the iterated model after the execution of the first round (a) and the second round (b). 90
5.3 The k-round protocol complex for 2 processes and $k \in \{1,2,3\}$. 92

6.1 General form of a protocol in the extended iterated model. 99
6.2 A graphical example of the Safe-Validity condition of the safe-consensus task. 101
6.3 General form of a protocol in the IFSC model 103
6.4 Part of the one round Protocol complex of a protocol in the IFSC model. The entire complex contains a countable number of copies of the subcomplex at the bottom. . 104
6.5 A $(n, n-1)$-set agreement protocol in the IFSC model 106
6.6 Subcomplexes of the first and second round protocol complexes of a protocol in the IFSC model. 109

7.1 A g-2coalitions-consensus protocol with one safe-consensus object. 115
7.2 The structure of the 4-consensus protocol using 2coalitions-consensus tasks. 116
7.3 An iterated consensus protocol using g-2coalitions-consensus objects. 118

8.1 Phase one of the Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}$ and box b_2. . 127
8.2 Phase two of the Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and b_2. 130
8.3 The Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and b_2. . . . 134
8.4 The path of the proof of Theorem 8.2.5, for $n = 7$ processes, the sets $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and the boxes b_1, b_2, b_3. 136

9.1 The combinatorial representation of the invocations of safe-consensus objects in a 4-process consensus protocol. 141
9.2 A subgraph of $J_{7,2}$ with $7-2=5$ vertices such that $\zeta^{7-2}(U) = \varnothing$. 142
9.3 The combinatorial interactions between paths of connected states of an iterated protocol $\mathcal{A}$ and the safe-consensus objects invoked by the processes. 148
9.4 The obstruction $\{\mathfrak{p}_0, \mathfrak{p}'_1, \ldots\}$ to reach consensus for the case $\nu_{\mathcal{A}'}(4,2) \leqslant 2$. 149

Part I:
Computable manifolds

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers. it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

Alan M. Turing.

1 Basic topology and computability theory

We begin our study of computable manifolds. In this chapter, we summarize many basic definitions, facts and technical details about topological manifolds and computability theory that will be used throughout this thesis. The reader can check the references for more complete information on these topics.

The chapter is organized as follows. Section 1.1 contains many basic definitions and results concerning topological manifolds (with boundary). In Section 1.2, we present basic definitions of standard computability theory. In Section 1.3, we give a short introduction to the *Type-2 theory of effectivity* (TTE for short). We first define Type-2 machines as a natural extension of standard Turing machines and define computable string functions in TTE. The next step will be to define standard topologies on Σ^* and Σ^ω, the sets of finite and infinite strings respectively and prove some basic results concerning the relationship between computability in TTE and topology. Finally, Multi-representations are defined and many useful basic results are proven.

1.1 Topology

We assume that the reader has a basic knowledge about general topology. Our references are [Arm83, Eng89, Mun00], but any standard textbook can be used as a reference.

Definitions 1.1.1. The power set of any set A will be denoted by $\mathcal{P}(A)$. A function $f\colon A \to B$ between the sets A, B, which is defined on a subset of A, is called a *partial function* and is denoted by $f\colon \subseteq A \to B$. When f is defined on the entire set A, f is called a *total function* and we omit the "$\subseteq$" symbol. The *identity* function on A is $1_A\colon A \to A$. For a topological space (X, τ), we denote by $\mathcal{A}$ the set of closed subsets of X and $\mathcal{K}$ will denote the set of compact subsets of X. A set $Z \subseteq X$ is a *G_δ-set* if Z is a countable intersection of open sets in X. The symbols $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ are used to represent the set of *natural numbers*; the set of *integers*; the set of *rational numbers* and the set of *real numbers* respectively. Let $\mathbb{R}^n = \{(x_1, \ldots, x_n) \mid x_i \in \mathbb{R}\}$ denote *euclidean space* of dimension $n \geqslant 0$. If (M, d) is a metric space with metric $d\colon M \times M \to \mathbb{R}$, then for any $x \in M$ and $0 < r \in \mathbb{R}$, let

$$B(x,r) = \{y \in M \mid d(x,y) < r\} \text{ and } \overline{B}(x,r) = \{y \in M \mid d(x,y) \leqslant r\}$$

denote the *open* and *closed ball* with center x and radius r.

We endow the euclidean space $\mathbb{R}^n$ with its standard topology, which is induced by the metric $d(x,y) = \|x - y\|$; $D^n = \overline{B}(0,1)$ is the *n-disk* (or *unit ball*), its topological boundary is known as the *$(n-1)$-sphere* $\mathbb{S}^{n-1}$, $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n \mid x_n \geqslant 0\}$ is called the *upper half space* and $\partial\mathbb{R}^n_+ = \{x \in \mathbb{R}^n \mid x_n = 0\}$ is the *boundary* of $\mathbb{R}^n_+$. All the previous subsets of $\mathbb{R}^n$ are equipped with subspace topology.

1.1.1 Topological manifolds

Our main objects of study are topological manifolds [KS77, Gau74, BC70, Fre82]. We first present one of the most common approaches to define them.

Definition 1.1.2. By a *topological n-manifold* (or just *manifold*) we mean a space M such that every point in M has an open neighborhood which is homeomorphic to an open subset of euclidean space $\mathbb{R}^n$. The integer n is called the *dimension* of the manifold and is denoted by $\dim M$.

Accordingly to this definition, a topological manifold is just a *locally euclidean space* in which all the points have the same local dimension. A manifold M of dimension n is also denoted by M^n. Most authors require their manifolds to satisfy further topological properties. At the least, manifolds are usually paracompact and Hausdorff. We will not assume any of these properties until we explicitly require them. It is easy to prove that the next result holds.

Proposition 1.1.3. *The following are equivalent for a space M.*

i) M is a topological manifold.

ii) Every point of M has a neighborhood homeomorphic to an open ball in $\mathbb{R}^n$.

iii) Every point of M has a neighborhood homeomorphic to $\mathbb{R}^n$ itself.

A neighborhood in M homeomorphic to an open ball in $\mathbb{R}^n$ is called an *euclidean ball*. The set of all euclidean balls in M form a basis for the topology of M. Being a topological manifold is a topological property. If M is a manifold and $f\colon M \to X$ is a homeomorphism, then X is also a topological manifold. We now present some examples.

Example 1.1.4. Euclidean space $\mathbb{R}^n$ is the prototypical n-manifold. Each point $x \in \mathbb{R}^n$ has the open neighborhood $U_x = \mathbb{R}^n$, which is clearly homeomorphic to $\mathbb{R}^n$.

Example 1.1.5. Any discrete space X is a 0-manifold. For $i \in X$, the open set $U_i = \{i\}$ is homeomorphic to 0-dimensional euclidean space.

Example 1.1.6. Any open subset $U \subseteq M$ of a n-manifold M is a n-manifold with the subspace topology. For let $x \in U$, then as M is a manifold, there exists an open neighborhood $V_x \subseteq M$ with $x \in V_x$ and V_x is homeomorphic to an open set of $\mathbb{R}^n$. Then $U \cap V_x$ is an open set in U (and M) and clearly $U \cap V_x$ is homeomorphic to an open set in $\mathbb{R}^n$, so that U is an (open) n-manifold.

Example 1.1.7. Let I be an index set and for each $\alpha \in I$, suppose that M_α is a manifold with $\dim M_\alpha = n$. Then the disjoint union $M = \bigcup_{\alpha \in I} M_\alpha$ is a n-manifold.

Lemma 1.1.8. *If M is a m-manifold and N is a n-manifold, the topological product $M \times N$ is a manifold such that* $\dim(M \times N) = m + n$.

Proof. Let $(x, y) \in M \times N$. We need to show that (x, y) has an open neighborhood on $M \times N$ which is homeomorphic to an open set of $\mathbb{R}^m \times \mathbb{R}^n = \mathbb{R}^{m+n}$. Because M and N are manifolds, there exist open neighborhoods $U \subseteq M, V \subseteq N$ of x and y respectively, such that $f\colon U \to \mathbb{R}^m, g\colon V \to \mathbb{R}^n$ are homeomorphisms of U and V onto open subsets of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively. Then the product

$$f \times g\colon U \times V \to \mathbb{R}^m \times \mathbb{R}^n$$

is an homeomorphism of $U \times V$ onto an open subset of $\mathbb{R}^m \times \mathbb{R}^n$. As $U \times V$ is an open neighborhood of (x, y) in $M \times N$ and (x, y) is any point, Definition 1.1.2 holds for $M \times N$, so that $M \times N$ is a topological manifold of dimension $m + n$. □

We will see more examples later.

1.1.2 Charts and atlases

We can also define topological manifolds without any explicit reference to some topology. This is done via the concepts of *chart* and *atlas*.

Definition 1.1.9. Let X be a set. A *coordinated chart* (or just *chart*) on X is a pair (φ, U) where $U \subseteq X$ and φ is a bijective function of U onto an open subset of $\mathbb{R}^m$. An n-dimensional *(topological) atlas* on X is a collection $\{(\varphi_i, U_i)\}_{i\in I}$ of coordinated charts on X such that

a) the sets U_i's cover X;

b) for each $i, j \in I$, $\varphi_i\left[U_i \cap U_j\right]$ and $\varphi_j\left[U_i \cap U_j\right]$ are open subsets of X;

c) Each map (called a *transition function*) $\varphi_j\varphi_i^{-1}\colon \varphi_i\left[U_i \cap U_j\right] \to \varphi_j\left[U_i \cap U_j\right]$ is a homeomorphism between open subsets of $\mathbb{R}^n$.

Suppose that $\Phi = \{(\varphi_i, U_i)\}_{i\in I}$ is a topological atlas on X. Define the set $\tau_\Phi \subset 2^X$ as follows: $V \in \tau_\Phi$ if and only if $\varphi_i\left[V \cap U_i\right]$ is open in $\mathbb{R}^n$ for each $i \in I$.

Proposition 1.1.10. *For any topological atlas Φ on X, τ_Φ is a topology in X.*

Proof. We first check that $\varnothing, X \in \tau_\Phi$. As it holds for any $i \in I$ that

$$\varphi_i\left[\varnothing\right] = \varnothing \quad \text{and} \quad \varphi_i\left[X \cap U_i\right] = \varphi_i\left[U_i\right],$$

then clearly $\varnothing, X \in \tau_\Phi$. To check that arbitrary unions and finite intersections of elements of τ_Φ are also members of τ_Φ, notice that as φ_i is bijective for any i, then

$$\varphi_i\left[\left(\textstyle\bigcup_l V_l\right) \cap U_i\right] = \textstyle\bigcup_l \varphi_i\left[V_l \cap U_i\right];$$

$$\varphi_i\left[\left(\textstyle\bigcap_l V_l\right) \cap U_i\right] = \textstyle\bigcap_l \varphi_i\left[V_l \cap U_i\right];$$

so that τ_Φ is closed under arbitrary unions and finite intersections. Therefore τ_Φ is a topology in X. □

Thus, an atlas Φ induces a topology τ_Φ in X. This topology has two very important properties:

- Each set U_i is open in X.
- Each map $\varphi_i\colon U_i \to \varphi_i\left[U_i\right]$ is a homeomorphism.

This says that the set X, equipped with the topology τ_Φ becomes a topological manifold as in Definition 1.1.2. However, a set X may have several different atlases defined on it and two such atlases need not induce the same topology on X. We need to be able to say when two distinct atlases induce the same topology on X. This can be achieved with the following concept.

Definition 1.1.11. Two n-dimensional atlases $\{(\varphi_i, U_i)\}, \{(\psi_j, V_j)\}$ on X are *compatible* if their union is an atlas on X.

This means that all the extra maps $\psi_j\varphi_i^{-1}$ must be homeomorphisms on their respective domains.

Lemma 1.1.12. *Let X be a set and Φ_1, Φ_2 two n-dimensional atlases on X. Then Φ_1 and Φ_2 are compatible if and only if they induce the same topology on X.*

Proof. ($\Rightarrow$) Suppose that $\Phi_1 = \{(\varphi_i, U_i)\}, \Phi_2 = \{(\psi_j, V_j)\}$ are compatible atlases on X and let $U \in \tau_{\Phi_1}$. By hypothesis, Φ_1 and Φ_2 are compatible and that means, by Definition 1.1.11, that $\Phi_1 \cup \Phi_2$ is an atlas on X. Using property b) of Definition 1.1.9, we can prove that $V_j \in \tau_{\Phi_1}$ for each j and as $U \in \tau_{\Phi_1}$, $U \cap V_j$ is open in the topology τ_{Φ_1} and that implies that $\varphi_i\big[U \cap V_j \cap U_i\big]$ is open in $\mathbb{R}^n$ for all i. By c) of Definition 1.1.9, the function

$$\psi_j\varphi_i^{-1} \colon \varphi_i\big[V_j \cap U_i\big] \to \psi_j\big[V_j \cap U_i\big]$$

is an homeomorphism, thus an open map, so that it sends the open set $\varphi_i\big[U \cap V_j \cap U_i\big]$ onto the open set $\psi_j\big[U \cap V_j \cap U_i\big]$. Now we know that $\psi_j\big[U \cap V_j\big] = \psi_j\big[(U \cap X) \cap V_j\big] = \psi_j\big[(U \cap (\bigcup_i U_i)) \cap V_j\big] = \psi_j\big[(\bigcup_i (U \cap U_i)) \cap V_j\big] = \psi_j\big[\bigcup_i (U \cap U_i \cap V_j)\big] = \bigcup_i \psi_j\big[U \cap U_i \cap V_j\big]$. In conclusion, $\psi_j\big[U \cap V_j\big] = \bigcup_i \psi_j\big[U \cap U_i \cap V_j\big]$ and this says that $\psi_j\big[U \cap V_j\big]$ is open in $\mathbb{R}^n$ for each j, so that $U \in \tau_{\Phi_2}$. We have proven that $\tau_{\Phi_1} \subseteq \tau_{\Phi_2}$ and to prove the other inclusion, the argument is symmetric. Hence, $\tau_{\Phi_1} = \tau_{\Phi_2}$ and the two atlases induce the same topology on X.

($\Leftarrow$) Since $\tau_{\Phi_1} = \tau_{\Phi_2}$, it is straightforward to prove that $\Phi_1 \cup \Phi_2$ is a n-dimensional atlas on X, so that Φ_1 and Φ_2 are compatible atlases, according to Definition 1.1.11. □

Thus, two atlases induce the same topological properties on a set X precisely when they are compatible. The compatibility of atlases is clearly an equivalence relation on the set[1] of all atlases of dimension n on X.

Definition 1.1.13. Let M be a set. An equivalence class $[\Phi]$ of atlases on M is called a *TOP structure* on M.

If M has a topology τ and an atlas defined on it, then the topology induced by the atlas is the same as τ if and only if each chart $\varphi\colon U \to \mathbb{R}^n$ is a homeomorphism in the topology τ. In summary, a TOP structure $[\Phi]$ on X is characterized by the topology induced by every atlas that belongs to $[\Phi]$.

Lemma 1.1.14. *A set M is a topological n-manifold if and only if it has a TOP structure of dimension n.*

Proof. ($\Rightarrow$) Assume that M is a n-manifold, according to Definition 1.1.2. For each point $x \in M$, let U_x be an open neighborhood of x in M, homeomorphic to an open set of $\mathbb{R}^n$, via the homeomorphism f_x. Then the set

$$\Phi = \{(f_x, U_x) \mid x \in M\},$$

is a topological atlas for M, with dimension n. Therefore M has a TOP structure induced by Φ.
($\Leftarrow$) If M has an atlas Φ, then this atlas induces a topology τ_Φ on M, and by the properties of Φ, M becomes a topological manifold with $\dim M = n$. □

[1]The collection of all n-dimensional atlases on X can be seen as a subset of $\mathcal{P}((\mathbb{R}^n)^X \times \mathcal{P}(X))$, by identifying a chart with a pair $(f\colon \subseteq X \to \mathbb{R}^n, U)$, where $U \subset X$ is the domain of f.

Notice that by the comment at the end of Definition 1.1.13, The topology τ of M as a topological manifold and the topology τ_Φ induced by the atlas Φ (and every atlas compatible with it) are the same.

Example 1.1.15. We construct an atlas for the 1-sphere $\mathbb{S}^1 \subset \mathbb{R}^2$. Let U_+, U_- be defined as

$$U_+ = \{(x,y) \in \mathbb{S}^1 \mid y > 0\} \quad \text{and} \quad U_- = \{(x,y) \in \mathbb{S}^1 \mid y < 0\},$$

and let $f_+ \colon U_+ \to \mathbb{R}, f_- \colon U_- \to \mathbb{R}$ be given by $f_+(x,y) = f_-(x,y) = x$, these two functions are injections of U_+, U_- onto $(-1,1) \subset \mathbb{R}$, thus the pairs $(f_+, U_+), (f_-, U_-)$ can be seen to be charts on $\mathbb{S}^1$, but still $U_+ \cup U_- \neq \mathbb{S}^1$. Now let

$$V_+ = \{(x,y) \in \mathbb{S}^1 \mid x > 0\} \quad \text{and} \quad V_- = \{(x,y) \in \mathbb{S}^1 \mid x < 0\},$$

and define $g_+ \colon V_+ \to \mathbb{R}, g_- \colon V_- \to \mathbb{R}$ as $g_+(x,y) = g_-(x,y) = y$. We claim that the set

$$\Phi = \{(f_+, U_+), (f_-, U_-), (g_+, V_+), (g_-, V_-)\}$$

is an 1-dimensional atlas on $\mathbb{S}^1$. We need to show that conditions a)-c) of Definition 1.1.9 are satisfied by Φ. Part a) is clearly true; to show part b), notice that the sets $\varphi[U \cap V]$ ($\varphi \in \{f_+, f_-, g_+, g_-\}; U, V \in \{U_+, U_-, V_+, V_-\}$) are

$$f_+[U_+ \cap U_-] = f_-[U_+ \cap U_-] = \varnothing;$$

$$f_+[U_+ \cap V_+] = g_+[U_+ \cap V_+] = (0,1);$$

$$f_+[U_+ \cap V_-] = (-1,0);$$

$$g_-[U_+ \cap V_-] = (0,1);$$

$$f_-[U_- \cap V_+] = (0,1);$$

$$g_+[U_- \cap V_+] = (-1,0);$$

$$f_-[U_- \cap V_-] = g_-[U_- \cap V_-] = (-1,0);$$

$$g_+[V_+ \cap V_-] = g_-[V_+ \cap V_-] = \varnothing;$$

and the sets $\varnothing, (0,1), (-1,0)$ are open sets of $\mathbb{R}$, thus b) is true for Φ. Finally, to prove part c), we only show that one of the transition maps is an homeomorphism, the other cases are similar. Take the composition $f_+ g_-^{-1} \colon g_-[U_+ \cap V_-] \to f_+[U_+ \cap V_-]$, a small computation shows that

$$f_+ g_-^{-1}(y) = -\sqrt{1-y^2}, \quad y \in (0,1),$$

and with the given parameters, it is easy to check that $f_+ g_-^{-1}$ is an homeomorphism of $g_-[U_+ \cap V_-] = (0,1)$ onto $f_+[U_+ \cap V_-] = (-1,0)$. Φ fulfills Definition 1.1.9, so that it is an atlas for $\mathbb{S}^1$.

Manifolds with boundary

The concept of manifold (Definition 1.1.2) does not include some simple sets like the n-disk D^n and $\mathbb{R}^n_+$ among others. We now extend the definition of topological manifold to encompass a broader range of topological spaces. Basically, we extend the concepts of charts and atlases given in Definition 1.1.9.

Definition 1.1.16. A *coordinated chart* on set X is a pair (φ, U) where $U \subseteq X$ and φ is a bijective function of U onto an open subset of $\mathbb{R}^m_+$. An n-dimensional *atlas* on X is a collection $\{(\varphi_i, U_i)\}_{i \in I}$ of coordinated charts on X such that

a) $X = \bigcup_i U_i$;

b) for each $i, j \in I$, $\varphi_i\big[U_i \cap U_j\big]$ is open in $\mathbb{R}^n_+$;

c) Each *transition function* $\varphi_j\varphi_i^{-1}\colon \varphi_i\big[U_i \cap U_j\big] \to \varphi_j\big[U_i \cap U_j\big]$ is a homeomorphism between open subsets of $\mathbb{R}^n_+$.

Everything that we said previously about atlases applies to the new atlases on a set X. Definition 1.1.11 of compatibility of atlases is also valid. A *n-dimension manifold with boundary* is a set M together with a equivalence class $[\Phi]$ of topological atlases (in the new sense).

Definition 1.1.17. Let $(M, [\Phi])$ be a n-manifold with boundary. We define the *boundary* ∂M and the *interior* $\operatorname{Int} M$ of M as follows.

$$\partial M = \{x \in M \mid (\exists(\varphi, U) \in \Phi)(x \in U \wedge \varphi(x) \in \partial\mathbb{R}^n_+)\};$$

$$\operatorname{Int} M = M - \partial M.$$

By the *Invariance of domain theorem*, ∂M is well defined, i.e., it does not depend on the chart. It is easy to see that ∂M is a manifold of dimension $n-1$ and $\operatorname{Int} M$ is a n-manifold, such that $\partial\partial M = \varnothing$ and $\partial\big(\operatorname{Int} M\big) = \varnothing$. A manifold N with $\partial N = \varnothing$ is called a *manifold without boundary* (or *with empty boundary*). Thus, Definition 1.1.2 correspond to manifolds with empty boundary. Here are some examples of manifolds with boundary.

- The half space $\mathbb{R}^n_+$ is a n-manifold with boundary $\partial\mathbb{R}^n_+$;
- The unit ball D^n is a manifold and its boundary is the $(n-1)$-sphere $\mathbb{S}^{n-1}$;
- The Möbius band is a 2-dimension manifold with boundary the 1-sphere;
- The cylinder $I \times \mathbb{S}^1$ is a 2-manifold with boundary the union of two disjoint circles.

The last example is a special case of a known result, which generalizes Lemma 1.1.8.

Lemma 1.1.18. *Let M, N be manifolds such that* $\dim M = m$, $\dim N = n$. *Then $M \times N$ is a $(m+n)$-manifold, such that*

$$\partial\big(M \times N\big) = (\partial M \times N) \cup (M \times \partial N).$$

More examples will come in the next chapters and there are even more in the references [BC70, Gau74, Mun00]. The most common way to define topological manifolds is using locally euclidean spaces. The definitions of charts and atlases are very handy in the context of differentiable and PL manifolds [BC70, Hud69], where the transition maps are required to have the additional properties of being C^k $(k \in \mathbb{N} \cup \{\infty\})$ and/or piecewise linear (PL) functions respectively. Atlases will be very useful in Chapter 3 for our work too, when we define computable manifolds using charts and atlases.

1.1.3 Submanifolds and embeddings

Manifolds which are subsets of other manifolds are an important tool to study topological manifolds and their properties. A topological manifold N is a *submanifold* of the manifold M if and only if $N \subset M$ and N is a subspace of M (that is, the topology of N is subspace topology). An easy example of a submanifold is any open set $U \subseteq M$ of any given manifold. The n-disk D^n is a compact submanifold of $\mathbb{R}^{n+1}$.

There are many facts [KS77] in the theory of manifolds (topological, differentiable and/or PL) which can be shown to be true using the well known result that every Hausdorff n-manifold M (of any kind) embeds in some high dimensional euclidean space $\mathbb{R}^q$, where q depends on n. Numerous versions of this embedding theorem exist, the big difference between them being the dimension of the space $\mathbb{R}^q$. It was proven by Whitney [Whi36] that if M is smooth (C^∞), then it embeds in $\mathbb{R}^{2n}$ and this is the best result possible. The same is true for the piecewise linear case using similar constructs to those used in the smooth case. If M has no additional structure, it can be embedded in $\mathbb{R}^{2n+1}$ and again, this is the lowest possible dimension for the euclidean space. This last result can be proven by means of dimension theory [HW48, Eng95, Mun00].

We will study submanifold and embeddings from the computational point of view in chapter 4, where we will prove an effective (i.e. computable) version of the next Theorem.

Theorem 1.1.19. *Let M^n be a compact Hausdorff manifold. Then for some q, there is an embedding $e\colon M \hookrightarrow \mathbb{R}^q$.*

A simple proof of Theorem 1.1.19 can be found in [Gau74]. We remark that this result is the "easy version" of the embedding theorem for topological manifolds.

1.2 Computability theory

In this section, we give a brief summary of definitions and terminology that we will be using. The reader that wishes to check a full introduction to basic computability theory and Type-2 theory of effectivity (TTE) can see the references [KW85, Koz97, Sip96, Coo04, Wei00, Wei08].

1.2.1 Basic notions of computability theory

To define a computable theory for continuous structures, our first step is to introduce the concept of *multi-function* [Wei00, Wei08]. This notion generalizes the definition of function and although multi-functions are very similar to set relations, they are not the same. The idea of multi-functions is fundamental in TTE, many computational problems themselves are more naturally expressed as multi-functions (i.e., multiple values assigned to a single element) rather than as standard functions.

Definition 1.2.1. Let A, B be sets. A *multi-function* from A to B is a triple $f = (A, B, R_f)$ such that $R_f \subseteq A \times B$ (this is the *graph* of f). We will denote it by $f\colon A \rightrightarrows B$. The inverse of f is the multi-function $f^{-1} \overset{\text{def}}{=} (B, A, R_{f^{-1}})$, where $R_{f^{-1}} \overset{\text{def}}{=} \{(b,a) \mid (a,b) \in R_f\} \subseteq B \times A$. For $X \subseteq A$, let $f[X] \overset{\text{def}}{=} \{b \in B \mid (\exists a \in X)(a,b) \in R_f\}$, $\operatorname{dom}(f) \overset{\text{def}}{=} f^{-1}[B]$ and $\operatorname{range}(f) \overset{\text{def}}{=} f[A]$. For $a \in A$, let $f(a) \overset{\text{def}}{=} f[\{a\}]$.

If it happens that for every $a \in A$, $f(a)$ contains at most one element, f can be treated as a usual partial function denoted by $f\colon \subseteq A \to B$. In contrast to relational composition, for multi-functions $f\colon A \rightrightarrows B$ and $g\colon B \rightrightarrows C$, we define the composition $g \circ f\colon A \rightrightarrows B$ by [Wei08, Section 3]

$$a \in \operatorname{dom}(g \circ f) \Leftrightarrow f(a) \subseteq \operatorname{dom}(g) \text{ and } g \circ f(a) \overset{\text{def}}{=} g\big[f(a)\big].$$

Definition 1.2.2. For a multi-function $f\colon X \rightrightarrows Y$ and $Z \subseteq Y$, define $f|^Z\colon X \rightrightarrows Z$ by $f|^Z(x) \overset{\text{def}}{=} f(x) \cap Z$ for all $x \in X$.

The multi-function $f|^Z$ of the previous definition is given in terms of the sets $f|^Z(x)$, if we want to define it in terms of Definition 1.2.1, then we have that $f|^Z = (X, Z, R_{f|^Z})$, where $R_{f|^Z} = \{(x,z) \in X \times Z \mid z \in f(x)\}$. Notice that when f is a function, then the multi-funcion $f|^Z$ from Definition 1.2.2 is simply the usual restriction of f to the set $f^{-1}\big[Z\big]$.

An *alphabet* is any non-empty finite set $\Sigma = \{a_1, \ldots, a_n\}$. We assume that any alphabet that we use contains at least the symbols 0, 1. We denote the set of finite words over Σ by Σ^*, the empty string by $\lambda \in \Sigma^*$ and with Σ^ω the set of infinite sequences $p\colon \mathbb{N} \to \Sigma$ over Σ. It is customary (and convenient) to treat an element p of Σ^ω as an "infinite word"

$$p(0)p(1)p(2)\cdots.$$

For $w, y \in \Sigma^*$ and $p \in \Sigma^\omega$, we say that *y starts with w* if $y = wz$ for some $z \in \Sigma^*$ (the notation "wz" means *concatenation*, that is, write the word z after w). Similarly, *p starts with w* if $p = wq$, where $q \in \Sigma^\omega$.

Definition 1.2.3. Let Σ be an alphabet. The *wrapping function* $\iota\colon \Sigma^* \to \Sigma^*$, is defined by

$$\iota(a_1 a_2 \cdots a_k) \overset{\text{def}}{=} 110a_1 0a_2 0 \cdots 0a_k 011.$$

The wrapping function is used for coding words in such a way that $\iota(u)$ and $\iota(v)$ cannot overlap. We will be using standard functions for finite or countable tupling on Σ^* and Σ^ω, all of them are denoted by $\langle\cdot\rangle$. The precise definitions of these functions are

$$\langle u_1, \ldots, u_n\rangle \overset{\text{def}}{=} \iota(u_1)\cdots\iota(u_n); \tag{1.1}$$

$$\langle u, p\rangle \overset{\text{def}}{=} \iota(u)p \tag{1.2}$$

$$\langle p, q\rangle \overset{\text{def}}{=} p(0)q(0)p(1)q(1)\cdots; \tag{1.3}$$

$$\langle p_0, p_1, \ldots\rangle\langle i, j\rangle \overset{\text{def}}{=} p_i(j); \tag{1.4}$$

for all $u, u_1, \ldots, u_n \in \Sigma^*$ and $p, q, p_i \in \Sigma^\omega$ $(i = 0, 1, \ldots)$. Each of the above functions is injective and even bijective, if the arguments are in Σ^ω. For $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, $u \ll w$ if and only if $\iota(u)$ is a subword of w.

1.2.2 Turing machines

We now define Turing machines, one of the most popular formal models of computation in standard computability theory [Wei87, HU90, Sip96, Koz97, Coo04]. Turing machines are used to define Type-2 machines in Section 1.3, which are a special type of Turing machines, with addition rules and some extensions.

Multi-tape Turing machines

There are many ways to define Turing machines, we will use one of the most useful definitions: The *multi-tape Turing machine*. We first give an informal description of what is a multi-tape Turing machine $\mathfrak{M}$ and the way it works (Figure 1.1). Intuitively, $\mathfrak{M}$ has $k \geqslant 1$ tapes, each tape is semi-infinite to the right and is divided into cells. The k tapes is the work space of the machine. Each head controls one tape of $\mathfrak{M}$ and each head can perform the following actions on the tape:

- Scan the contents of the current cell in the tape;
- modify the symbol of the current cell in the tape;
- move to the left or right of the tape to scan another cell. Also, the head can stay in the current cell without moving.

In one or more tapes, $\mathfrak{M}$ receives input strings, which represent the input data which $\mathfrak{M}$ uses to perform its work. When the machine starts the execution of its program, it is in a "start" state s, with the heads of the input tapes scanning the first symbol of each of the input strings. In each step of $\mathfrak{M}$, depending on the current state of the machine and the symbols scanned by the k read/write heads, $\mathfrak{M}$ changes the current state, writes a symbol on the current cell of each tape, and finally $\mathfrak{M}$ determines for each read/write head if it must be moved to scan the contents of another cell or if the head must stay in the same cell. The full program of $\mathfrak{M}$ is specified by means of a transition function. When the machine enters into a special state t, called the "accept" state, it stops all actions and the computation is successful. On the other hand, if $\mathfrak{M}$ enters into another special state r, called the "reject" state, it stops but with failure. It can also happen that $\mathfrak{M}$ never enters one of the states t, r, in this case, we say that $\mathfrak{M}$ is in a infinite loop.

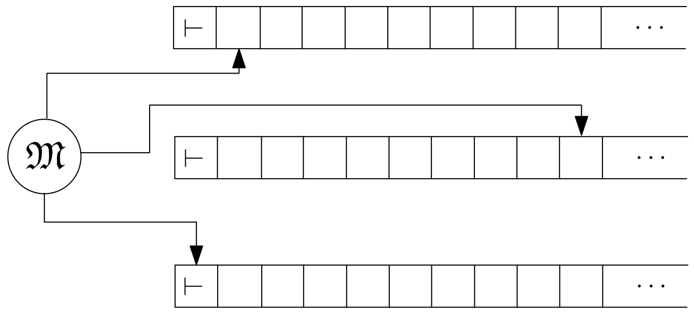


Figure 1.1: A Turing machine with three tapes.

The formal definition of multi-tapes Turing machines is as follows.

Definition 1.2.4. A *deterministic multi-tape Turing machine* is a tuple

$$\mathfrak{M} = (Q, \Sigma, \Gamma, \delta, s, t, r),$$

where

- Q is a finite non-empty set; elements of Q are called *states*;
- Σ is the input alphabet;

- Γ is the work alphabet, where $\Sigma \cup \{\sqcup\} \subseteq \Gamma$;
- $\delta \colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$, is the transition function (the program);
- $s \in Q$, is the start state;
- $t \in Q$, is the accept state;
- $r \in Q$, is the reject state, $r \neq t$.

The symbols L, R and S stand for left, right and stay, respectively.

Intuitively, a transition of the form $\delta(q, a_{i1}, a_{i2}, \ldots, a_{ik}) = (p, b_{i1}, b_{i2}, \ldots, b_{ik}, D_1, D_2, \ldots, D_k)$ has the following meaning: If the machine $\mathfrak{M}$ is in the state q, scanning the symbol a_{i1} on the first tape, the symbol a_{i2} on the second tape, ..., the symbol a_{ik} on the k tape, then the machine must switch the current state to p, write on the first tape the symbol b_{i1} (replacing a_{i1}), on the second tape the symbol $b_{i2}, \ldots,$ on the k tape the symbol b_{ik} (replacing a_{ik}) and finally, move the head of the first tape in direction D_1, move the head of the second tape in direction $D_2, \ldots,$ move the head of tape k in direction D_k. When the machine enters in one of the two special states t or r, it is not allowed to do any more transitions or state changes, we say that $\mathfrak{M}$ *halts*. The alphabet Γ always contains a special symbol "$\sqcup$", called the *blank symbol*, which is used to indicate that the corresponding tape cell is empty.

Two way infinite tapes

We have given the definition of Turing machine using one way infinite tapes. If we modify Definition 1.2.4 to allow the use of two way infinite tapes, it can be proved that we do not add any extra power to the model [Sip96, Koz97]. Thus, we can build Turing machines with one way infinite tapes, two way infinite tapes, and machines with both types of tapes, all these variations of the Turing machine model have the same computational power. In Section 1.3, we use this fact to specify Type-2 machines as multi-tape Turing machines with both types of tapes.

1.2.3 Tarsky's decision method for the elementary algebra of the reals

In order to prove some results in the next chapters, we will be using a celebrated result from A. Tarsky [Tar48]. We first give the definition of *elementary expression*.

Definition 1.2.5. An *elementary expression* in the algebra of the real numbers, is an expression build with the following objects:

- Variables over the real numbers.
- Constants $c \in \mathbb{N}$.
- The symbols $+, -, \cdot, \div$ which denote sum, subtraction, multiplication and division of real numbers respectively.
- The symbols $>, =$ that denote the relations "greater than" and "equal to" respectively, of real numbers.
- The logic connectives $\vee$ (disjunction), $\wedge$ (conjunction), $\neg$ (negation) and $\Rightarrow$ (implication).

- The universal ($\forall$) and existential ($\exists$) quantifiers.

Notice that in general, it is impossible to say something about sets of real numbers with elementary expressions. Although we can give expressions for sets like $\{5, 7, 10000\}$ (with the elementary expression $x = 5 \vee x = 7 \vee x = 10000$), it is impossible to write down with an elementary expression the statement "x is an integer", that is, the expression $x \in \mathbb{Z}$ is not elementary, neither is the following expression about integer equations

$$(\exists x, y, z \in \mathbb{Z})(x^3 + y^3 = z^3).$$

If such expressions were to be elementary, that would imply that all sentences of elementary number theory, are elementary expressions, and that would mean that Theorem 1.2.6 below is false [Tar48], thus the previous expressions about elements of $\mathbb{Z}$ are not elementary. However, all basic properties of order, sum and multiplication of the field $\mathbb{R}$ can be expressed as elementary expressions. Also, many useful properties of the ring $\mathbb{R}[x_1, \ldots, x_n]$ ($n \geqslant 1$) can be given with elementary expressions. We are now ready to introduce Tarsky's result.

Theorem 1.2.6. *There exists an algorithm to decide, given an elementary expression in the algebra of the real numbers, whether it is true or false.*

More details on Tarsky's method can be found in [Tar48]. From now on, in this thesis, whenever we need to apply the Tarski's decision method, we assume that we have an appropriate encoding of polynomial functions with rational coefficients as strings of Σ^*.

1.3 Type-2 theory of effectivity

It is time to begin our short introduction to the computability theory of Type-2 Theory of Effectivity (TTE for short). The main tool is the Type-2 machines, which are some sort of generalized Turing machines, which can operate on infinite strings. This will be our first definition of TTE. Later, we define computability with Type-2 machines for finite and infinite strings, multi-functions defined on strings and sets, generalizing the same notions presented in ordinary computability theory. Another important tool of TTE is the idea of representations and notations, which are (roughly speaking) codifications of abstract objects with strings of some alphabet, these encodings are used to transfer computability notions of TTE to arbitrary sets. We give some of the most useful results regarding representations of computable functions.

1.3.1 Type-2 machines and computable string functions

For $k > 0$ and $Y_0, \ldots, Y_k \in \{\Sigma^*, \Sigma^\omega\}$, let $Y = \prod_{i=1} Y_i$. We define the computable functions $f :\subseteq Y \to Y_0$ by means of Turing machines with k one-way input tapes, finitely many work tapes and a single one-way output tape. Figure 1.2 shows such a Turing machine.

Definition 1.3.1. A Type-2 machine is a multi-tape Turing machine $\mathfrak{M} = (Q, \Sigma, \Gamma, \delta, s, t, r)$ with $N + 1$ tapes, which satisfies the following additional rules:

- The tapes $1, \ldots, k$ are the *input tapes* of $\mathfrak{M}$ ($k < N$);
- the tapes $k + 1, \ldots, N$ are the *work tapes* of $\mathfrak{M}$;

- the tape 0 is the *output tape* of $\mathfrak{M}$
- the tuple $(Y_1, \ldots, Y_k, Y_0)$ $(Y_i \in \{\Sigma^*, \Sigma^\omega\})$ is called the *type specification* of $\mathfrak{M}$. For each $i = 1, \ldots, k$, Y_i denotes the type of input strings for input tape i and Y_0 specifies the type of strings for the output tape.

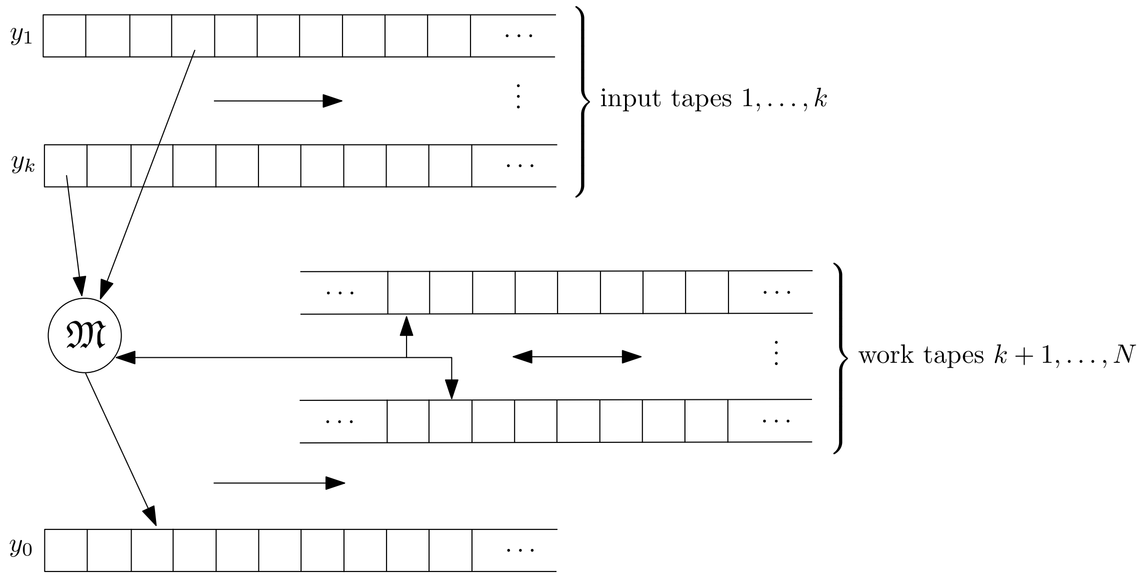


Figure 1.2: A Type-2 machine computing $y_0 = f_{\mathfrak{M}}(y_1, \ldots, y_k)$.

The machine $\mathfrak{M}$ must satisfy the following restrictions:

- The head of an input tape cannot be moved to the left, neither can modify a symbol of the input string.
- The head of the output tape can move only to the right, writing a new symbol in each move.
- Any symbol written to the output tape must be an element of Σ.

The restrictions of Type-2 machines guarantee that the input tapes are one-way read-only tapes and that on the output tape, only symbols from Σ can be written and no written symbol can be erased (one-way output). For the rest of the section, set $Y = \prod_i^k Y_i$. We now define the string function $f\colon \subseteq Y \to Y_0$ computed by a Type-2 machine $\mathfrak{M}$.

Definition 1.3.2. The initial tape configuration for input $(y_1, ..., y_k) \in Y$ is as follows: for each input tape i, the (finite or infinite) sequence $y_i \in Y_i$ is placed on the tape immediately to the right of the head, all other tape cells contain the symbol $\sqcup$; on all the other tapes, all cells contain the symbol $\sqcup$. For all $y_0 \in Y_0$, $(y_1, ..., y_k) \in Y$, we define the function $f_{\mathfrak{M}}\colon \subseteq Y \to Y_0$ computed by $\mathfrak{M}$ as:

Case $Y_0 = \Sigma^*$:
$f_{\mathfrak{M}}(y_1, \ldots, y_k) = y_0 \in \Sigma^*$, if and only if $\mathfrak{M}$ halts (on its accept state) on input $(y_1, \ldots, y_k)$ with y_0 on the output tape.

Case $Y_0 = \Sigma^\omega$:
$f_{\mathfrak{M}}(y_1, \dots, y_k) = y_0 \in \Sigma^\omega$ if and only if $\mathfrak{M}$ computes forever on input $(y_1, ..., y_k)$ and writes y_0 on the output tape, i.e., the machine $\mathfrak{M}$ never halts and keeps computing the infinite string y_0 symbol by symbol.

We call a string function $f\colon \subseteq Y \to Y_0$ *computable*, if it is computed by some Type-2 machine $\mathfrak{M}$.

In case that $\mathfrak{M}$ halts on the reject state or $\mathfrak{M}$ never halts, but writes only finitely many symbols on the output tape, then we say that $f_{\mathfrak{M}}(y_1, \dots, y_k)$ is undefined, that is, $(y_1, \dots, y_k) \notin \operatorname{dom} f_{\mathfrak{M}}$. We do not use the "partial" results of such computations in our definition of semantics. Since Type-2 machines are essentially Turing machines, they are as realistic and powerful as Turing machines. Clearly, infinite inputs or outputs do not exist and infinite computations cannot be finished in reality. But finite computations on finite initial parts of inputs producing finite initial parts of the outputs can be realized on physical devices as long as enough time and memory are available.

Of course, Type-2 machines can be simulated by digital computers. Therefore, infinite computations of Type-2 machines can be *approximated* by finite physical computations with arbitrary precision. The restriction to one-way output guarantees that any partial output of a finite initial part of a computation cannot be erased in the future and, therefore, is final. For this reason, models of computation with two-way output would not be very useful.

Definition 1.3.2 includes the standard definition of computable string functions (choose $Y_0 = \dots = Y_k = \Sigma^*$). We define computable elements of Σ^* and Σ^ω as follows:

Definition 1.3.3. Let $w \in \Sigma^*$, $p \in \Sigma^\omega$ and $y = (y_1, y_2, \dots, y_k) \in Y$.

1. The finite string w is computable.
2. The sequence p is computable, if and only if $p = f(\lambda)$ for some computable function $f\colon \Sigma^* \to \Sigma^\omega$
3. The tuple y is computable, if and only if each component y_i is computable.

We illustrate the definitions by several examples.

Example 1.3.4. A constant function $f\colon Y \to Y_0$ is computable, if and only if its value $c \in Y_0$ is computable. Also, every projection $pr_i\colon Y \to Y_i$ is computable.

Example 1.3.5. Define $f\colon \subseteq \Sigma^\omega \to \Sigma^*$ by,

$$f(p) = \begin{cases} 1 & \text{if } p \neq 0^\omega, \\ \text{undef} & \text{otherwise.} \end{cases}$$

There is a Type-2 machine $\mathfrak{M}$ which reads the infinite input string $a_0 a_1 \ldots \in \Sigma^\omega$ and writes 1 and halts, as soon as some i with $a_i \neq 0$ has been found. Therefore $\mathfrak{M}$ computes f.

Example 1.3.6. The function $f\colon \Sigma^\omega \to \Sigma^*$ defined by,

$$f(p) = \begin{cases} 1 & \text{if } p \neq 0^\omega, \\ 0 & \text{otherwise.} \end{cases}$$

is not computable. To see this, assume that some Type-2 machine $\mathfrak{M}$ computes f. On input $0^\omega = 00\ldots$, $\mathfrak{M}$ halts with result 0 within t steps for some t. Then $\mathfrak{M}$ halts with result 0 also on input $q \overset{\text{def}}{=} 0^t 1^\omega \in \Sigma^\omega$. But $f(q) = 1$, a contradiction. Thus f is not computed by any Type-2 machine.

We will be using throughout this thesis the tupling functions defined earlier in Section 1.2.1. As in ordinary recursion theory, these functions are used to represent functions with several arguments by functions with a single argument. The next result will be applied in an implicit way in many proofs and arguments.

Theorem 1.3.7 ([Wei00]). *Let Σ be an alphabet. The following statements are true:*

- *Each of the tupling functions defined in Equations (1.1), (1.2) and (1.3) is computable, and every projection of its inverse is computable.*
- *For the infinite tupling function of (1.4), the function $(0^i, q) \mapsto pr_i \circ \langle\rangle^{-1}$ (uniform projection of the inverse) is computable.*

More examples and preliminary results concerning TTE can be found in [Wei87, Wei00]. Computability on Σ^* and Σ^ω can be reduced to computability on $\{0,1\}^*$ and $\{0,1\}^\omega$, respectively, by encoding the symbols of the alphabet $\Sigma^* = \{a_1, \ldots, a_n\}$ as follows: $a_i \mapsto 1^{i-1}0$ for $i < n$ and $a_n \mapsto 1^n$. The computable functions on Σ^* and Σ^ω are closed under composition and even under programming [Wei00, Wei08, TW11]. The composition of computable functions has a computable extension. With the following definition, the notions of *computable enumerable* and *computable* from ordinary computability theory [Koz97, Coo04] is generalized to TTE.

Definition 1.3.8. Let $W, Z \subseteq Y$, the set W is called *computable enumerable (c.e.) in* Z if $W = Z \cap \operatorname{dom} f$ for some computable function $f\colon \subseteq Y \to \Sigma^*$. W is called *computable* if W and $Z - W$ are both c.e. When $Z = Y$, we omit "in Z".

For $k = 1$ and $Z = Y_1 = \Sigma^*$, we obtain the ordinary definition of c.e. and computable sets [Koz97, Coo04]. The following characterization of c.e. subsets of Σ^* will be used later in Chapter 3, its proof can be found in standard textbooks, e.g., [Coo04].

Lemma 1.3.9. *Let $A \subseteq \Sigma^*$. Then A is a c.e. set if and only if $A = \varnothing$ or there exists a total computable function $h\colon \Sigma^* \to \Sigma^*$ such that* $\operatorname{range} h = A$. *Moreover, if A is infinite, then h can be chosen to be injective.*

Finally, Definition 1.3.10 resembles Oracle Turing machines from the standard theory [Sip96, Koz97, Coo04].

Definition 1.3.10. Let $\mathfrak{M}$ be a Type-2 machine with $k+1$ input tapes and let $f_{\mathfrak{M}^p}(y) \overset{\text{def}}{=} f_{\mathfrak{M}}(p, y)$, where p is fixed and $(p, y) \in Y_1 \times Y_2 \times \cdots \times Y_k \times Y_{k+1}$. We say that $\mathfrak{M}$ is an *oracle Type-2 machine* and $f_{\mathfrak{M}^p}$ is the function computed by the machine $\mathfrak{M}$ with the *oracle p*.

Remark. In some proofs or arguments, we will need to show that some multi-functions are computed by Type-2 machines and prove some sets to be computably enumerable. Giving the full specification of such machines can be a rather long and tedious task. To avoid this, we will only write these Type-2 machines using a simple description or with pseudocode, which will contain the main steps to compute the function we are proving to be computable and of course, we justify the computability of each one of these steps. In this way, our proofs will not become needlessly long and boring. We can content ourselves with a overall description or pseudocode because, as we have said before, the multi-functions computed by Type-2 machines are closed under programming, this is stated by the main results of [Wei08, TW11].

1.3.2 Topologies for Σ^* and Σ^ω

In order to define computability on topological spaces, it is convenient to define topologies for the sets Σ^* and Σ^ω.

Definition 1.3.11. We equip Σ^* with its discrete topology τ_*. For each $z \in \Sigma^*$, we define the set

$$z\Sigma^\omega = \{p \in \Sigma^\omega \mid p \text{ starts with } z\} \subset \Sigma^\omega.$$

Lemma 1.3.12. *The set $\beta_C = \{z\Sigma^\omega \mid z \in \Sigma^*\}$ is a basis of a topology on Σ^ω.*

Proof. We will show that β_C satisfies the two properties:

1. Σ^ω is covered by the elements of β_C.
2. For $z_1\Sigma^\omega, z_2\Sigma^\omega \in \beta_C$ and $y \in z_1\Sigma^\omega \cap z_2\Sigma^\omega$, there exists $z_3\Sigma^\omega \in \beta_C$ such that $y \in z_3\Sigma^\omega \subset z_1\Sigma^\omega \cap z_2\Sigma^\omega$.

Property 1 is clearly true, as for any $p \in \Sigma^\omega$, $p = \lambda p$, so that $p \in \lambda\Sigma^\omega$ and this implies that $\Sigma^\omega \subseteq \lambda\Sigma^\omega$ and $\lambda\Sigma^\omega \in \beta_C$. For property 2, let $z_1, z_2 \in \Sigma^*$, given the sets $z_1\Sigma^\omega, z_2\Sigma^\omega \in \beta_C$, we have that

$$z_1\Sigma^\omega \cap z_2\Sigma^\omega = \begin{cases} z_1\Sigma^\omega & \text{if } z_2 \text{ starts with } z_1, \\ z_2\Sigma^\omega & \text{if } z_1 \text{ starts with } z_2, \\ \varnothing & \text{otherwise.} \end{cases}$$

Thus, for any $y \in z_1\Sigma^\omega \cap z_2\Sigma^\omega$, $y \in z_1\Sigma^\omega = z_1\Sigma^\omega \cap z_2\Sigma^\omega$ or $y \in z_2\Sigma^\omega = z_1\Sigma^\omega \cap z_2\Sigma^\omega$. In any case, β_C fulfills property 2. Therefore, β_C is a base of a topology on Σ^ω. □

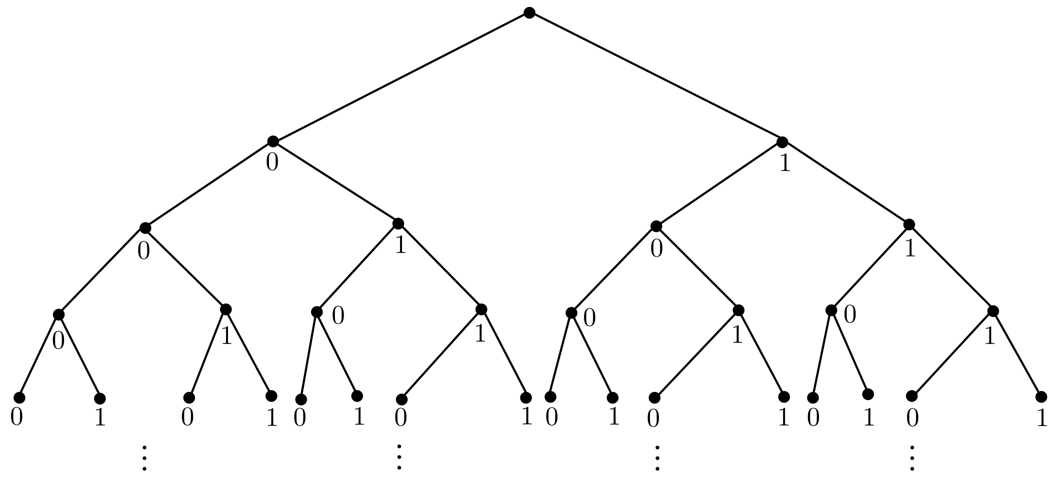


Figure 1.3: The topological space $(\{0,1\}^\omega, \tau_C)$.

The topology generated on Σ^ω by the base β_C is called the *Cantor topology* on Σ^ω and is denoted by τ_C. It can be checked that τ_C is defined as

$$\tau_C = \{A\Sigma^\omega \mid A \subseteq \Sigma^*\},$$

where $A\Sigma^\omega = \{p \in \Sigma^\omega \mid (\exists y \in A)(p \text{ starts with } y)\} = \bigcup_{z\in A} z\Sigma^\omega$. The set Σ^ω of infinite sequences over Σ can be visualized by a tree where every $p \in \Sigma^\omega$ is represented by an infinite descending path from the root. Figure 1.3 shows this tree for the alphabet $\Sigma = \{0,1\}$ with two elements. A basis element $z\Sigma^\omega \in \beta_C$ is represented by a full subtree and an open subset of Σ^ω is represented by a set of full subtrees. Figure 1.4 shows four subtrees representing the open set $001\Sigma^\omega \cup 011\Sigma^\omega \cup 10\Sigma^\omega \cup 11\Sigma^\omega$.

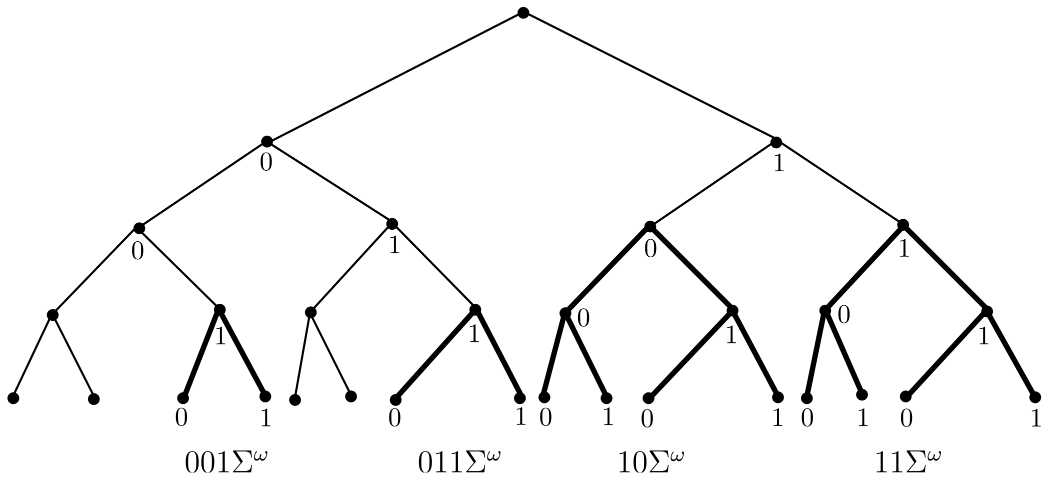


Figure 1.4: The subtrees representing the open set $001\Sigma^\omega \cup 011\Sigma^\omega \cup 10\Sigma^\omega \cup 11\Sigma^\omega$.

Definition 1.3.13. For the product $Y = \prod_i^k Y_i$ with $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ for each $i = 1,\dots,k$, we equip Y with the product topology τ_Y generated by the topologies of $Y_1,\dots,Y_k$. Also, consider the set $\beta_Y = \{y \circ Y \mid y = (y_1,\dots,y_k) \text{ and } y_i \in \Sigma^*\}$, where $(y_1,\dots,y_k) \circ Y = U_1 \times \cdots \times U_k$ with

$$U_i = \begin{cases} \{y_i\} & \text{if } Y_i = \Sigma^*, \\ y_i\Sigma^\omega & \text{if } Y_i = \Sigma^\omega. \end{cases}$$

It is not hard to see that β_Y is a basis for τ_Y. With the topologies defined on Σ^* and Σ^ω, we can prove a result which gives a nice connection between topology and computability in TTE theory.

Theorem 1.3.14. *Let $f\colon \subseteq Y \to Y_0$ be a computable function. Then f is continuous.*

Proof. Let $\mathfrak{M}$ be a Type-2 machine computing f. Assume that $y = (y_1,\dots,y_k) \in \operatorname{dom} f \subseteq Y = Y_1 \times \cdots \times Y_k$. We have two cases.

Case $Y_0 = \Sigma^*$. It suffices to show that $f^{-1}\big[\{w\}\big]$ is open for every baseelement $\{w\}$ of τ_*. Assume $f(y_1,\dots,y_k) = w$. Since the machine $\mathfrak{M}$ halts on this input, during its computation from every input tape i with $Y_i = \Sigma^\omega$, it can read only a prefix $u_i \in \Sigma^*$ of $y_i \in \Sigma^\omega$. If $Y_i = \Sigma^*$, then define $u_i = y_i$. Then $y \in (u_1,\dots,u_k) \circ Y$ and $f\big[(u_1,\dots,u_k) \circ Y\big] \subseteq \{w\}$. The set $(u_1,\dots,u_k) \circ Y$ is an open neighborhood of $y = (y_1,\dots,y_k)$ contained in $f^{-1}\big[\{w\}\big]$. Therefore, $f^{-1}\big[\{w\}\big]$ is open.

Case $Y_0 = \Sigma^\omega$. It is enough to show that $f^{-1}\big[w\Sigma^\omega\big]$ is open for every base element $w\Sigma^\omega$ of τ_C. Assume $f(y_1,\dots,y_k) \in w\Sigma^\omega$. Since on this input the machine $\mathfrak{M}$ needs only finitely many computaton steps for producing the prefix w of the result, during this computation, from every input tape i with $Y_i = \Sigma^\omega$, it can read only a prefix $u_i \in \Sigma^*$ of $y_i \in \Sigma^\omega$. If $Y_i = \Sigma^*$, then define $u_i = y_i$. Then $y \in (u_1, \ldots, u_k) \circ Y$ and $f\big[(u_1, \ldots, u_k) \circ Y\big] \subseteq w\Sigma^\omega$. The set $(u_1, \ldots, u_k) \circ Y$ is an open neighborhood of $(y_1, \ldots, y_k)$ contained in $f^{-1}\big[w\Sigma^\omega\big]$. Therefore, $f^{-1}\big[w\Sigma^\omega\big]$ is open.

We have proven all the required cases, thus f is a continuous function from Y to Y_0. □

The domains of computable functions have nice topological properties, this is stated in the following

Lemma 1.3.15. *The following statements are satisfied:*

1. *If $f\colon \subseteq Y \to \Sigma^*$ is computable, then* $\operatorname{dom} f$ *is an open set.*
2. *If $f\colon \subseteq Y \to \Sigma^\omega$ is computable, then* $\operatorname{dom} f$ *is a G_δ-set.*

Proof. 1. In the proof of Theorem 1.3.14, we have shown that every element $(y_1, \ldots, y_k) \in \operatorname{dom} f$ has an open neighborhood $(u_1, \ldots, u_k) \circ Y \subseteq \operatorname{dom} f$.Therefore, $\operatorname{dom} f$ is open. 2. Let $\mathfrak{M}$ be a Type-2 machine which computes f. Then, for each $n \in \mathbb{N}$, there is a machine $\mathfrak{M}_n$ which on input $y \in Y$ halts, if $\mathfrak{M}$ on input y writes at least n symbols on the output tape, and does not halt otherwise. By Property 1, $\operatorname{dom} f_{\mathfrak{M}_n}$ is open for each n, therefore,

$$\operatorname{dom} f = \bigcap_{i=0}^{\infty} \operatorname{dom} f_{\mathfrak{M}_i}$$

is a G_δ-set. □

It can also be proven [Wei00] that a function $f\colon \subseteq Y \to Y_0$ is continuous if and only if for some Type-2 machine $\mathfrak{M}$ and some oracle $p \in \Sigma^\omega$, $f_{\mathfrak{M}^p}$ extends f. Finally, $W \subseteq Y$ is open if and only if for some Type-2 machine $\mathfrak{M}$ with output set Σ^* and some oracle p, $W = \operatorname{dom}\big(f_{\mathfrak{M}^p}\big)$. Regarding Definition 1.3.8, there is a striking similarity to topological concepts, where "c.e." corresponds to "open": By definition,

1. X is *open* in Z, if and only if $X = U \cap Z$ for some open set $U \subseteq Y$;
2. X is *open* and *closed in* Z, if and only if X and $Z - X$ are open in Z.

1.3.3 Multi-representations and representations

In TTE, computability on finite or infinite sequences of symbols is transferred to other sets by means of notations and representations, where elements of Σ^* or Σ^ω are used as "concrete name" of abstract objects. We will need the more general concept of realization via *multi-representations* (see [Wei08, Section 6] for a detailed discussion, and also [Sch03, WG09]).

Definition 1.3.16. A *multi-representation* of a set M is a surjective multi-function $\gamma\colon \subseteq Z \rightrightarrows M$ where $Z \in \{\Sigma^*, \Sigma^\omega\}$. If γ is single-valued, it is called a *representation* and if in addition, $Z = \Sigma^*$, then $\gamma\colon \subseteq \Sigma^* \to M$ is called a *notation* of the set M. Given $m \in M$, an element $z \in Z$ such that $m \in \gamma(z)$ is called a *name* of m.

Example 1.3.17. For any alphabet Σ, we define $\nu_{\Sigma^*} \overset{\text{def}}{=} 1_{\Sigma^*}$, the trivial notation for Σ^*.

Example 1.3.18. Let $\Sigma = \{0,1\}$ and $\nu_{\mathbb{N}} \colon \subseteq \Sigma^* \to \mathbb{N}$ be the usual binary notation of natural numbers. Using the standard bijection[2] $\langle \cdot,\cdot \rangle \colon \mathbb{N}^2 \to \mathbb{N}$ defined by

$$\langle x, y \rangle \overset{\text{def}}{=} \frac{(x+y)(x+y+1)}{2} + y$$

and the derived bijections $\langle \cdots \rangle \colon \mathbb{N}^{k+1} \to \mathbb{N}$ defined by $\langle x_1, \ldots, x_k, x_{k+1} \rangle \overset{\text{def}}{=} \langle \langle x_1, \ldots, x_k \rangle, x_{k+1} \rangle$ for $k \geqslant 2$, we define notations $\nu_{\mathbb{N}^n} \colon \Sigma^* \to \mathbb{N}^n$ for the set $\mathbb{N}^n$ ($n \geqslant 1$) by

$$\nu_{\mathbb{N}^n}(w) = (x_1, \ldots, x_n) \iff \nu_{\mathbb{N}}(w) = \langle x_1, \ldots, x_n \rangle. \qquad \boxed{1.5}$$

Example 1.3.19. By the previous example with $k = 3$, the bijection $\langle \cdots \rangle \colon \mathbb{N}^3 \to \mathbb{N}$ can be used to build a surjection $g_Q \colon \mathbb{N} \to \mathbb{Q}$ defined by

$$g_Q(\langle i, j, k \rangle) \overset{\text{def}}{=} \frac{k+1}{i-j}.$$

We define a notation $\nu_{\mathbb{Q}} \colon \subseteq \Sigma^* \to \mathbb{Q}$ of the set $\mathbb{Q}$ such that $\operatorname{dom}(\nu_{\mathbb{Q}}) = \operatorname{dom}(\nu_{\mathbb{N}})$ and $\nu_{\mathbb{Q}}$ makes the following diagram

$$\begin{tikzcd}
\mathbb{N} \arrow[r, "g_Q"] & \mathbb{Q} \\
\Sigma^* \arrow[u, "\nu_{\mathbb{N}}"] \arrow[ur, "\nu_{\mathbb{Q}}"'] &
\end{tikzcd}$$

commute, i.e., $\nu_{\mathbb{Q}}(w) = q \iff q = g_Q(\nu_{\mathbb{N}}(w))$. In words, $\nu_{\mathbb{Q}}(w) = q$ if and only if w represents a natural number $n_q = \langle i, j, k \rangle$ such that $q = \frac{k+1}{i-j}$.

Example 1.3.20. Let $g_Q^n \colon \mathbb{N}^n \to \mathbb{Q}^n$ be defined by $g_Q^n(m_1, \ldots, m_n) = (g_Q(m_1), \ldots, g_Q(m_n))$. Generalizing Example 1.3.19, we define notations $\nu_{\mathbb{Q}^n} \colon \subseteq \Sigma^* \to \mathbb{Q}^n$ for $n \geqslant 1$ ($\mathbb{Q}^1 = \mathbb{Q}$) by the following rules: $\operatorname{dom}(\nu_{\mathbb{Q}^n}) = \operatorname{dom}(\nu_{\mathbb{N}^n})$ and the diagram

$$\begin{tikzcd}
\Sigma^* \arrow[r, "\nu_{\mathbb{Q}^n}"] \arrow[d, "\nu_{\mathbb{N}^n}"'] & \mathbb{Q}^n \\
\mathbb{N}^n \arrow[ur, "g_Q^n"'] &
\end{tikzcd}$$

is commutative, or equivalently,

$$\nu_{\mathbb{Q}^n}(w) = (q_1, \ldots, q_n) \iff \nu_{\mathbb{N}^n}(w) = (m_1, \ldots, m_n) \text{ and } (\forall i \in \{1, \ldots, n\})(q_i = g_Q(m_i)). \qquad \boxed{1.6}$$

In words, $\nu_{\mathbb{Q}^n}(w) = (q_1, \ldots, q_n)$ if and only if w represents a tuple $(m_1, \ldots, m_n) \in \mathbb{N}^n$ such that the number m_i encodes the rational q_i, in the way described in Example 1.3.19.

[2]Notice how we use the same notation $\langle \cdot,\cdot \rangle$ to denote the bijections between $\mathbb{N}^k$ and $\mathbb{N}$ and the tupling functions between strings from Σ^* and Σ^ω (see Equations $\boxed{1.1}$,$\boxed{1.4}$, $\boxed{1.2}$ and $\boxed{1.3}$). This is customary in computability theory and one can deduce from the context the precise meaning of the notation $\langle \cdot,\cdot \rangle$.

Example 1.3.21. The usual decimal representation of $\mathbb{R}$ can be defined precisely so that it is a representation $\rho_{10}\colon \subseteq \Sigma^\omega \to \mathbb{R}$ in the sense of Definition 1.3.16, Let $\Sigma = \{-, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The set $\operatorname{dom}\rho_{10}$ contains all infinite strings from Σ^ω which represent real numbers in decimal form. For example, some elements of $\operatorname{dom}\rho_{10}$ are

$\rho_{10}(1.414\ldots) = \sqrt{2};$

$\rho_{10}(1.000\ldots) = 1;$

$\rho_{10}(3.141516\ldots) = \pi;$

$\rho_{10}(0.5000\ldots) = \frac{1}{2};$

$\rho_{10}(-0.999\ldots) = -1;$

whereas the following examples are not valid elements of $\operatorname{dom}\rho_{10}$

$-00..00111\ldots;$

$2.000.001.22\ldots;$

etc. Similarly, one can define representations to any other base $b \in \mathbb{N}$ with $b \geqslant 2$.

Example 1.3.22. We define the *Cauchy representation* $\rho\colon \subseteq \Sigma^\omega \to \mathbb{R}$ of the real numbers [Wei00] as follows: Let $\Sigma = \{0, 1, \#\}$, the domain of ρ contains all infinite strings in Σ^ω such that

$$\#w_1\#w_2\#w_3\#\cdots \quad (w_i \in \operatorname{dom}\nu_{\mathbb{Q}}),$$

that is, a string $\#w_1\#w_2\#\cdots \in \operatorname{dom}\rho$ encodes an infinite sequence $(\nu_{\mathbb{Q}}(w_i))_{i=0}^\infty$ of rational numbers (the # symbol is just a delimiter) and this sequence must converge to a real number $x \in \mathbb{R}$, in symbols

$$\rho(\#w_1\#w_2\#\cdots) = x \Longleftrightarrow d(x, \nu_{\mathbb{Q}}(w_i)) < 2^{-i}, \text{ for all } i \in \mathbb{N}. \tag{1.7}$$

For example, if $\nu_{\mathbb{Q}}(z_m) = \frac{1}{2^m}$ for all $m \geqslant 1$, then $\rho(\#z_1\#z_2\#\cdots) = 0$. If $\nu_{\mathbb{Q}}(y_k) = \sum_{j=1}^k \frac{3}{10^k}$ with $k \geqslant 1$, then $\rho(\#y_1\#y_2\#\cdots) = 0.333\cdots$. The Cauchy representation can be easily generalized to a representation $\rho^n\colon \subseteq \Sigma^\omega \to \mathbb{R}^n$ of n-dimensional euclidean space for all $n \geq 0$.

Mathematical examples of multi-representations will be given later. If $\nu\colon \subseteq \Sigma^* \to X$ is a notation and $\delta\colon \subseteq \Sigma^\omega \rightrightarrows Y$ is a multi-representation for X and Y respectively, sometimes we will denote $\nu(w) \in X$ and $\delta(p) \subseteq Y$ by ν_w and δ_p (see Section 1.3.4).

Definition 1.3.23. For multi-representations $\gamma_i\colon Y_i \rightrightarrows M_i$ $(0 \leqslant i \leqslant n)$, let $Y = \prod_{i=1}^n Y_i$, $M = \prod_{i=1}^n M_i$ and $\gamma\colon Y \rightrightarrows M$, $\gamma(y_1, \ldots, y_n) = \prod_i \gamma_i(y_i)$. A partial function $h\colon \subseteq Y \to Y_0$ *realizes* the multi-function $f\colon M \rightrightarrows M_0$ if

$$f(x) \cap \gamma_0 \circ h(y) \neq \varnothing \text{ whenever } x \in \gamma(y) \text{ and } f(x) \neq \varnothing. \tag{1.8}$$

Equation $\boxed{1.8}$ means that $h(y)$ is a name of some $z \in f(x)$ if y is a name of $x \in \operatorname{dom} f$. This situation is depicted in Figure 1.5. If $f\colon \subseteq M \to M_0$ is single-valued, then $h(y)$ is a name of $f(x)$ if y is a name of $x \in \operatorname{dom} f$. If only the representations are single-valued, then $\gamma_0 \circ h(y) \in f(x)$ when $\gamma(y) = x$. For the case when f and the representations are all functions, the situation is the usual equation given by the following commutative diagram

$$\begin{tikzcd} M \arrow[r, "f"] & M_0 \\ Y \arrow[u, "\gamma"] \arrow[r, "h"] & Y_0 \arrow[u, "\gamma_0"] \end{tikzcd} \quad f \circ \gamma(y) = \gamma_0 \circ h(y).$$

With realizations, we can induce computability and continuity concepts on the represented sets M, M_0.

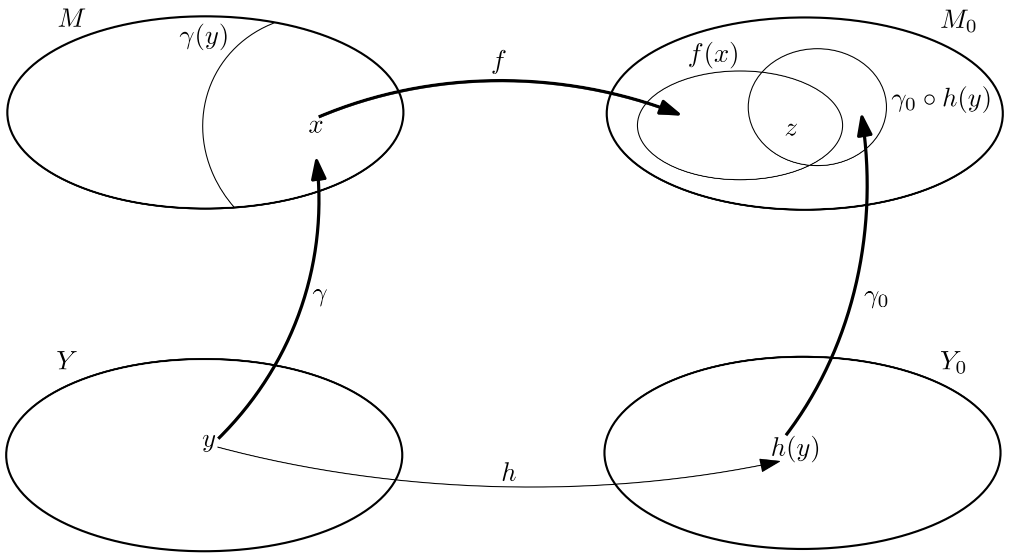


Figure 1.5: The function $h\colon \subseteq Y \to Y_0$ realizes the multi-function $f\colon \subseteq M \rightrightarrows M_0$.

Definition 1.3.24. Let $f, \gamma_0, \ldots, \gamma_n$ be as above.

1. The multi-function f is called $(\gamma_1, \ldots, \gamma_n, \gamma_0)$-*computable* if it has a computable realization.
2. f is called $(\gamma_1, \ldots, \gamma_n, \gamma_0)$-*continuous* if it has a continuous realization.

If the multi-representations are fixed, we occasionally say that f is *relatively continuous (relatively computable)*. The relatively continuous (computable) functions are closed under composition , even more, they are closed under GOTO-programming with indirect addressing [Wei08, TW11].

Computable enumerable sets and reducibility

Now we extend the definition of computable elements of Σ^*, Σ^ω, and c.e. sets [Wei00] to arbitrary sets via multi-representations.

Definition 1.3.25. With γ_i and γ from Definition 1.3.23, a point $x \in M_1$ is γ_1-computable if and only if $x \in \gamma_1(p)$ for some computable $p \in \operatorname{dom} \gamma_1$. A set $S \subseteq M$ is $(\gamma_1, \ldots, \gamma_n)$-c.e. if there is a c.e. set $W \subseteq Y$ such that

$$x \in S \Leftrightarrow y \in W$$

for all x, y with $x \in \gamma(y)$.

Therefore, $S \subseteq M$ is $(\gamma_1, \ldots, \gamma_n)$-c.e. if and only if there is a Type-2 machine that halts on input $y \in \operatorname{dom} \gamma$ if and only if y is a name of some $x \in S$.

Definition 1.3.26. Let γ_0, γ_1 be as in Definition 1.3.23. We say that γ_1 is *reducible* to γ_0 (denoted by $\gamma_1 \leq \gamma_0$) if $M_1 \subseteq M_0$ and the inclusion $i_{M_1} \colon M_1 \hookrightarrow M_0$ is (γ_1, γ_0)-computable.

This means that some computable function h translates γ_1-names to γ_0-names, that is, $\gamma_1(p) \subseteq \gamma_0 \circ h(p)$. When γ_0 and γ_1 are singled-valued, the situation is the usual equation $\gamma_1(p) = \gamma_0 \circ h(p)$. Reducibility can be used to define computable equivalences between multi-representations.

Definition 1.3.27. The multi-representations γ_0, γ_1 are called *computably equivalent* if $\gamma_1 \leq \gamma_0 \wedge \gamma_0 \leq \gamma_1$. When such an equivalence exists, we denote it by $\gamma_1 \sim \gamma_0$.

Two multi-representations induce the same computability on a set if and only if they are computably equivalent. For $X \subseteq M_1$, if X is γ_0-c.e. and $\gamma_1 \leq \gamma_0$, then X is γ_1-c.e.

Open sets and continuous reducibility

As each one of the sets Σ^*, Σ^ω has a topology, we can take the open sets in this topological spaces and give similar definitions to those given previously for c.e. sets.

Definition 1.3.28. Let γ_i and γ be as in Definition 1.3.23. A set $S \subseteq M$ is $(\gamma_1, \ldots, \gamma_n)$-open if there is an open set $W \subseteq Y$ such that

$$x \in S \Leftrightarrow y \in W$$

for all x, y with $x \in \gamma(y)$.

Thus, S is $(\gamma_1, \ldots, \gamma_n)$-open if and only if there is a Type-2 machine with oracle that halts on input $w \in \operatorname{dom} \gamma$ if and only if w is a name of some $z \in S$. Notice that $\gamma\gamma^{-1}[S] = S$ if S is γ-open.

Definition 1.3.29. Let γ_0, γ_1 be as in Definition 1.3.23. We say that γ_1 is *continuously reducible* to γ_0 (denoted by $\gamma_1 \leq_t \gamma_0$) if $M_1 \subseteq M_0$ and the inclusion $i_{M_1} \colon M_1 \hookrightarrow M_0$ is (γ_1, γ_0)-continuous.

This means that some continuous function h translates γ_1-names to γ_0-names, that is, $\gamma_1(p) \subseteq \gamma_0 \circ h(p)$. As we have done for the case of computable reducibility, continuous reducibility can be used to define continuous equivalences between multi-representations.

Definition 1.3.30. The multi-representations γ_0, γ_1 are called *continuously equivalent* if $\gamma_1 \leq_t \gamma_0 \wedge \gamma_0 \leq_t \gamma_1$. When such an equivalence exists, we denote it by $\gamma_1 \sim_t \gamma_0$.

Two multi-representations induce the same continuity on a set if and only if they are continuously equivalent. For $X \subseteq M_1$, if X is γ_0-open and $\gamma_1 \leq_t \gamma_0$, then X is γ_1-open.

Some useful notations and multirepresentations

Definition 1.3.31. Let $\gamma_i \colon Y_i \rightrightarrows M_i$ ($Y_i \in \{\Sigma^*, \Sigma^\omega\}, i = 1, 2$) be multi-representations. The multi-representation $[\gamma_1, \gamma_2]$ of the product $M_1 \times M_2$ is defined by

$$[\gamma_1, \gamma_2] \langle y_1, y_2 \rangle \overset{\text{def}}{=} \gamma_1(y_1) \times \gamma_2(y_2).$$

Since the function $(x_1, x_2) \mapsto (x_1, x_2)$ is $(\gamma_1, \gamma_2, [\gamma_1, \gamma_2])$-computable and the projection $(x_1, x_2) \mapsto x_i$ is $([\gamma_1, \gamma_2], \gamma_i)$-computable $(i = 1, 2)$, a multi-function $g \colon M_1 \times M_2 \rightrightarrows M_0$ is $(\gamma_1, \gamma_2, \gamma_0)$-computable if and only if g is $([\gamma_1, \gamma_2], \gamma_0)$-computable. A set is (γ_1, γ_2)-open if and only if it is $[\gamma_1, \gamma_2]$-open, etc.

In this thesis, we will be using the canonical notations given in [WG09] of finite and of countable subsets of a given set.

Definition 1.3.32. For the notation $\mu \colon \subseteq \Sigma^* \to M$ and representation $\gamma \colon \subseteq \Sigma^\omega \to Y$, define the notation μ^{fin} and representations μ^{cs}, γ^{fin}, γ^{cs} of finite or countable subsets of M and Y as follows. Consider $w \in \Sigma^*, q, p_i \in \Sigma^\omega$ and $a_i \in \Sigma$, $i = 0, 1, \ldots$, then

$$\mu^{\text{fin}}(w) = W \iff \begin{cases} (\forall v \ll w)(v \in \operatorname{dom}(\mu)), \\ W = \{\mu(v) \mid v \ll w\}; \end{cases} \tag{1.9}$$

$$\mu^{\text{cs}}(q) = W \iff \begin{cases} (\forall v \ll q)(v \in \operatorname{dom}(\mu)), \\ W = \{\mu(v) \mid v \ll q\}; \end{cases} \tag{1.10}$$

$$\gamma^{\text{fin}}(q) = Z \iff \begin{cases} (\exists n)(\exists p_1, \ldots, p_n \in \operatorname{dom}(\gamma)) \\ q = \langle 1^n, p_1, \ldots, p_n \rangle, \\ Z = \{\gamma(p_1), \ldots, \gamma(p_n)\}; \end{cases} \tag{1.11}$$

$$\gamma^{\text{cs}}\langle a_0 p_0, a_1 p_1, \cdots \rangle = Z \iff \begin{cases} (\forall i)(a_i = 0 \Rightarrow p_i \in \operatorname{dom}(\gamma)) \\ Z = \{\gamma(p_i) \mid a_i = 0\}. \end{cases} \tag{1.12}$$

If $a_i \neq 0$ for all i, then $\gamma^{\text{cs}}\langle a_0 p_0, a_1 p_1, \cdots \rangle = \varnothing$.

The next lemma is easy to prove and it will be applied in proofs without further mentioning.

Lemma 1.3.33 ([WG09]). *For the notation μ and notations or representations β, γ,*

1. *The set* $\operatorname{dom}(\mu^{fin})$ *is computable if* $\operatorname{dom}(\mu)$ *is computable,*
2. *The function $(x, y) \mapsto \{x, y\}$ is $(\gamma, \gamma, \gamma^{fin})$-computable,*
3. *$\gamma' \leq \gamma^{fin} \leq \gamma^{cs}$, where $\gamma'(w) = \{\gamma(w)\}$,*
4. *$\beta^{fin} \leq \gamma^{fin}$ and $\beta^{cs} \leq \gamma^{cs}$ if $\beta \leq \gamma$.*

Remark. In some cases, the notation μ^{fin} will be used to give abstract names to finite unions or intersections of a collection $\mathcal{C}$ of subsets of a set X, where $\mu \colon \Sigma^* \to \mathcal{C}$. To avoid confusion about which set operation we refer to with the notation μ^{fin}, we will denote μ^{fin} as $\bigcup_{\text{fin}} \mu$ when we want to encode the finite union of elements of $\mathcal{C}$ and when we want to use μ^{fin} to describe finite intersections of the members of $\mathcal{C}$, we write $\bigcap_{\text{fin}} \mu$ instead of μ^{fin}.

1.3.4 The smn and utm properties

To obtain a powerful and elegant theory, in [Wei00], representations $\eta^{ab}\colon \Sigma^\omega \to F^{ab}$ of sets of continuous functions $f\colon \subseteq \Sigma^a \to \Sigma^b$ are introduced for $a, b \in \{*, \omega\}$. The definitions of the represented sets F^{ab} are given as follows.

$$F^{**} = \{f \mid f\colon \subseteq \Sigma^* \to \Sigma^*\};$$

$$F^{*\omega} = \{f \mid f\colon \subseteq \Sigma^* \to \Sigma^\omega\};$$

$$F^{\omega *} = \{f \mid f\colon \subseteq \Sigma^\omega \to \Sigma^* \text{ is continuous and } \operatorname{dom} f \text{ is open}\};$$

$$F^{\omega\omega} = \{f \mid f\colon \subseteq \Sigma^\omega \to \Sigma^\omega \text{ is continuous and } \operatorname{dom} f \text{ is a } G_\delta\text{-set }\}.$$

Notice that the sets $F^{**}, F^{*\omega}$ contain all continuous partial functions of each kind, because as every subset of Σ^* is open, every function $f\colon \subseteq \Sigma^* \to \Sigma^b$ is continuous. Also, each set F^{*b} ($b \in \{*, \omega\}$) has the same cardinality as Σ^ω or $\mathbb{R}$. Therefore these sets can be represented.

For the case of continuous functions of the form $f\colon \Sigma^\omega \to \Sigma^b$, things are a bit complicated. Let $g\colon \Sigma^\omega \to \Sigma^b$ be a constant function. Clearly, g and its restrictions are all continuous functions and the set of all these restrictions has the cardinality of the power set of Σ^ω. Thus, the set of all partial continuous functions from Σ^ω to Σ^b cannot be represented. The problem is solved by considering only continuous functions with natural domains, which in some sense represent all partial continuous functions.

Remember that by Lemma 1.3.15, the domain of every computable function from $F^{\omega *}$ is open and the domain of every computable function from $F^{\omega\omega}$ is a G_δ-set. The following theorem may serve as further justification for the definitions of $F^{\omega *}$ and $F^{\omega\omega}$. Roughly, it says that the sets $F^{\omega *}$ and $F^{\omega\omega}$ represent essentially all partial continuous functions $f\colon \subseteq \Sigma^\omega \to \Sigma^*$ and $f\colon \subseteq \Sigma^\omega \to \Sigma^\omega$, respectively. The second part of the theorem is a special case of a well known extension theorem for continuous functions on metric spaces [Kur66].

Theorem 1.3.34 ([Wei00])**.** *The following statements hold.*

1. *Every continuous partial function $f\colon \subseteq \Sigma^\omega \to \Sigma^*$ has an extension in $F^{\omega *}$.*
2. *Every continuous partial function $f\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ has an extension in $F^{\omega\omega}$.*

This is why the set $F^{\omega *}$ contains only partial continuous functions with open domains and $F^{\omega\omega}$ contains only partial continuous functions with domains equal to G_δ-sets.

The representations η^{ab} have two important and useful properties. First of all, computable string functions have computable names[3] [Wei00].

Lemma 1.3.35. *For $a, b \in \{*, \omega\}$, a partial function $f\colon \subseteq \Sigma^a \to \Sigma^b$ is computable if and only if $\eta^{ab}(p) = \eta^{ab}_p = f$ for some computable $p \in \operatorname{dom}(\eta^{ab})$.*

Also, each representation η^{ab} satisfies the following [Wei00, Theorem 2.3.13]

Theorem 1.3.36. *For each $a, b \in \{*, \omega\}$, the next statements are satisfied:*

utm *There is a computable (universal) function $U\colon \subseteq \Sigma^\omega \times \Sigma^a \to \Sigma^b$ with $U(p, x) = \eta^{ab}_p(x)$.*

[3]Remember that we denote $\eta^{ab}(p)$ as η^{ab}_p.

smn *For every computable function $f\colon \subseteq \Sigma^\omega \times \Sigma^a \to \Sigma^b$ there is a total computable function $r\colon \Sigma^\omega \to \Sigma^\omega$ such that $f(p,x) = \eta^{ab}_{r(p)}(x)$.*

The representations η^{ab} can be used to build multi-representations of multi-functions of represented sets. For multi-representations $\gamma_1\colon \subseteq \Sigma^a \rightrightarrows M_1$ and $\gamma_2\colon \subseteq \Sigma^b \rightrightarrows M_1$, $a, b \in \{*, \omega\}$, a multi-representation denoted by $[\gamma_1 \rightrightarrows \gamma_2]$ of the (γ_1, γ_2)-continuous multi-functions $f\colon M_1 \to M_2$ is defined by:

$$f \in [\gamma_1 \rightrightarrows \gamma_2]\,(p) \iff \eta^{ab}_p \text{realizes } f \text{ with respect to } \gamma_1 \text{ and } \gamma_2.$$

See [Wei08]. The restriction of $[\gamma_1 \rightrightarrows \gamma_2]$ to the single-valued functions is called $\left[\gamma_1 \to_p \gamma_2\right]$ in [Wei08] or $[\gamma_1 \to \gamma_2]_{set}$ in [Wei00]. The restriction of $[\gamma_1 \rightrightarrows \gamma_2]$ to the total (γ_1, γ_2)-continuous functions is denoted by $[\gamma_1 \to \gamma_2]$ ([Sch02, Wei08, KW85, Wei00, Sch02]).

The generalization of Theorem 1.3.36 to represented sets is known as the type conversion theorem, [Wei00, Theorem 3.3.15] for single-valued representations and total functions and [Wei08, Theorem 33] as the most general version.

The reader that wishes to learn or review in more depth the concepts and results about computability and Type-2 theory of effectivity, is encouraged to consult the references [Koz97, Sip96, Coo04, Wei00, Wei08]. In the next chapter, we will use TTE to develop a theory of computability for general T_0 topological spaces with countable bases.

2 Computable topology

In this chapter, we introduce the basic concepts of computable topology that we need in order to define computable manifolds. We use the framework of Type-2 theory of effectivity to introduce computability in T_0 topological spaces with countable bases. We will define *Computable topological spaces* and study many basic constructions that we will be using in later chapters to define computable manifolds. Also, we will study computability of continuous functions between computable topological spaces and a constructive version of the Hausdorff property.

The chapter is organized as follows. In Section 2.1, we introduce the definition of *Computable topological space* [WG09] and present many of their important properties. We also define computable subspaces and products of computable topological spaces. In Section 2.2, *Computable functions* are defined by using representations of continuous functions between topological spaces. Also, *computable homeomorphisms* and *embeddings* are studied. In Section 2.3, we present a computable version of the *Hausdorff* (T_2) property. This property will be used in Chapter 4 to prove our last result. Our main references for this chapter are [Wei00, WG09, Wei10].

2.1 Computability inside topological spaces

With the study of computable real numbers and functions, came Computable Analysis, which provided a solid theory of computability for basic continuous structures, like euclidean spaces, metric and normed vector spaces [Wei00, Bra98, PER89]. All these developments were the starting point of what is called *Computable Topology*, where computability is introduced in more general topological spaces. In this section, we give the concept of computable topological space and workout many examples. After that, we present some simple (but very useful) ways to construct computable topological spaces from simple computable assumptions and we give the computable versions of the concepts of subspaces and products of computable topological spaces.

2.1.1 Computable T_0 spaces

Roughly speaking, a computable topological space is a T_0-space (X, τ) in which a base $\beta \subseteq \tau$ is provided with an encoding with strings from Σ^*, such that the set of all valid strings that encode elements of β is computable and under the given encoding, the intersection of base elements is computable. The formal way to say this is as follows.

Definition 2.1.1 ([GWX08, WG09]). An *effective topological space* is a 4-tuple $\mathbf{X} = (X, \tau, \beta, \nu)$ such that (X, τ) is a T_0-space and $\nu\colon \subseteq \Sigma^* \to \beta$ is a notation of a base $\beta \subseteq \tau$. $\mathbf{X}$ is a *computable* topological space if $\operatorname{dom}\nu$ is computable and there exists a c.e. set $S \subseteq (\operatorname{dom}\nu)^3$ such that

$$\nu(u) \cap \nu(v) = \bigcup\{\nu(w) \mid (u,v,w) \in S\} \text{ for all } u,v,w \in \operatorname{dom}\nu, \qquad \boxed{2.1}$$

Since the base β has a notation, it must be countable. Topological spaces with countable bases are called *second countable* [Eng89, Mun00]. Equation $\boxed{2.1}$ says that in a computable topological space the intersection of base elements is computable[1]. For every effective topological space there is some (not necessarily c.e.) set S such that $\boxed{2.1}$ is true.

Example 2.1.2 (*Computable euclidean space*)**.** Let $\Sigma = \{0,1\}$. Define $\mathbf{R}^n \overset{\text{def}}{=} (\mathbb{R}^n, \tau^n, \beta^n, \mu^n)$ such that τ^n is the usual topology on $\mathbb{R}^n$ and $\mu^n : \Sigma^* \to \beta^n$ is a notation of the set β^n of all open balls in $\mathbb{R}^n$ with rational radii and center, defined as follows: $\operatorname{dom}(\mu^n) = \operatorname{dom}(\nu_{\mathbb{Q}^{n+1}})$ and

$$\mu^n(w) = B(q,r) \iff \nu_{\mathbb{Q}^{n+1}}(w) = (q_1,\dots,q_n,r), \quad q = (q_1,\dots,q_n),$$

(See Example 1.3.20). The inclusion of a rational ball $B(q_1,r_1)$ in the intersection of two rational balls $B(q_2,r_2) \cap B(q_3,r_3)$ can be decided by a Turing machine (uniformly on the input parameters), because it can be expressed as

$$(\forall x)(d(x,q_1) < r_1 \Rightarrow d(x,q_2) < r_2 \wedge d(x,q_3) < r_3),$$

and this predicate can be reformulated as an elementary expression, so that we can apply Theorem 1.2.6 and we can use this fact to prove that $\boxed{2.1}$ of Definition 2.1.1 is satisfied. Thus, $\mathbf{R}^n$ is a computable topological space. Since this computable space is very important throughout all the thesis, we fix once and for all the notation used in this example to denote the elements of $\mathbf{R}^n$. When $n = 1$, $\mathbf{R}^1 = \mathbf{R}$ is the *computable real line*.

Example 2.1.3. Let λ be a notation of a set A. Define an effective topological space $\mathbf{A} = (A, \tau, \beta, \nu)$ by $\nu(w) \overset{\text{def}}{=} \{\lambda(w)\}$, $\beta \overset{\text{def}}{=} \operatorname{range}(\nu)$ and τ is the discrete topology on A. If the set

$$\{(u,v) \mid u,v \in \operatorname{dom}\lambda, \lambda(u) = \lambda(v)\}$$

is c.e., then we can easily show that the set $S = \{(u,v,w) \mid \lambda(u) = \lambda(v) = \lambda(w)\} \subseteq (\operatorname{dom}\nu)^3$ is c.e. and S fulfills $\boxed{2.1}$ for $\mathbf{A}$. By Definition 2.1.1, $\mathbf{A}$ is a computable topological space.

Example 2.1.4. Let $\Sigma = \{0,1\}$ and $\nu_{\mathbb{Q}} : \subseteq \Sigma^* \to \mathbb{Q}$ be the notation for the set $\mathbb{Q}$ from Example 1.3.19. Define the effective space $\mathbf{R}_< \overset{\text{def}}{=} (\mathbb{R}, \tau_<, \beta_<, \nu_<)$ as follows: $\operatorname{dom}\nu_< = \operatorname{dom}\nu_{\mathbb{Q}}$, $\nu_<(w) \overset{\text{def}}{=} (\nu_{\mathbb{Q}}(w), \infty)$, $\beta_< \overset{\text{def}}{=} \operatorname{range}(\nu_<)$ and $\tau_< \overset{\text{def}}{=} \{(x,\infty) \mid x \in \mathbb{R}\} \cup \{\varnothing, \mathbb{R}\}$. Since we have that the expression

$$\nu_<(w) \subseteq \nu_<(u_1) \cap \nu_<(u_2)$$

is equivalent to the elementary expression

$$(\forall x)(\nu_{\mathbb{Q}}(w) < x \Rightarrow \nu_{\mathbb{Q}}(u_1) < x \wedge \nu_{\mathbb{Q}}(u_2) < x),$$

then we can use Theorem 1.2.6 to find a c.e. set $S \subseteq (\operatorname{dom}\nu_<)^3$ which satisfies Definition 2.1.1. Therefore $\mathbf{R}_<$ is a computable topological space.

[1]Using the representation θ of open sets given in Definition 2.1.6, the function which maps two base elements to their intersection is (ν,ν,θ)-computable (see Definition 1.3.24).

Example 2.1.5. Let $\mathbf{E} = (\Sigma^\omega, \tau_C, \beta_C, \nu_C)$, where $\beta_C = \{w\Sigma^\omega \mid w \in \Sigma^*\}$ and $\nu_C \colon \Sigma^* \to \beta_C$ is defined by $\nu_C(w) = w\Sigma^\omega$. The topology τ_C is the Cantor topology generated by the base β_C (Section 1.3.2). Now, $w\Sigma^\omega \subseteq u\Sigma^\omega \cap v\Sigma^\omega \Leftrightarrow w$ is a prefix of both strings u and v and this can be checked algorithmically, thus we can enumerate a set $S \subseteq (\operatorname{dom}\nu_C)^3$ which satisfies Equation $\boxed{2.1}$, so that the effective space $\mathbf{E}$ is a computable topological space.

More examples of computable topological spaces can be found in [Wei00] and [WG09]. The definition of computable topological space allows us to define several multi-representations of the points of X and many classes of subsets (open, closed, compact, etc) [WG09]. All these representations are an important piece to define computability inside computable spaces. We will use the notations $\bigcup_{\text{fin}} \nu$ and $\bigcap_{\text{fin}} \nu$ of the finite unions and finite intersections respectively, of the base sets of a computable topological space $\mathbf{X} = (X, \tau, \beta, \nu)$, see Definition 1.3.32 for details on the definition of the notations $\bigcup_{\text{fin}} \nu$ and $\bigcap_{\text{fin}} \nu$. As usual, we assume that $\bigcap \varnothing = X$ and $\bigcup \varnothing = \varnothing$.

Definition 2.1.6 (Positive information representations)**.** Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be an effective topological space. Define a representation $\delta \colon \subseteq \Sigma^\omega \to X$ of X, a representation $\theta \colon \subseteq \Sigma^\omega \to \tau$ of the set of open sets, a representation $\psi \colon \subseteq \Sigma^\omega \to \mathcal{A}$ of the set of closed sets, a multi-representation $\widetilde{\psi} \colon \subseteq \Sigma^\omega \rightrightarrows \mathcal{P}(X)$ of the power set and a multi-representation $\kappa \colon \subseteq \Sigma^\omega \rightrightarrows \mathcal{K}$ of the set of compact subsets of X as follows:

$$x = \delta(p) \Longleftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu \text{ and } x \in \nu(w)), \tag{2.2}$$

$$W = \theta(p) \Longleftrightarrow \begin{cases} w \ll p \Rightarrow w \in \operatorname{dom}\nu, \\ W = \bigcup_{w \ll p} \nu(w), \end{cases} \tag{2.3}$$

$$A = \psi(p) \Longleftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu \text{ and } A \cap \nu(w) \neq \varnothing), \tag{2.4}$$

$$B \in \widetilde{\psi}(p) \Longleftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu \text{ and } B \cap \nu(w) \neq \varnothing), \tag{2.5}$$

$$K \in \kappa(p) \Longleftrightarrow (\forall z \in \Sigma^*)(z \ll p \Leftrightarrow z \in \operatorname{dom}\big(\bigcup_{\text{fin}} \nu\big) \text{ and } K \subseteq \bigcup_{\text{fin}} \nu(z)), \tag{2.6}$$

The previous representations give us information about the represented object, that is, about its contents. We now define representations that complement the previous ones, in the sense that these representations can say something about the "complements" of the objects that the representations δ, θ and ψ are encoding.

Definition 2.1.7 (Negative information representations)**.** Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be an effective topological space. Define a representation $\delta^- \colon \subseteq \Sigma^\omega \to X$ of the points, a representation $\theta^- \colon \subseteq \Sigma^\omega \to \tau$ of the set of open sets, a representation $\psi^- \colon \subseteq \Sigma^\omega \to \mathcal{A}$ of the set of closed sets by

$$\delta^-(p) = x \Longleftrightarrow \theta(p) = X - \overline{\{x\}}, \tag{2.7}$$

$$\theta^-(p) \overset{\text{def}}{=} X - \psi(p), \tag{2.8}$$

$$\psi^-(p) \overset{\text{def}}{=} X - \theta(p). \tag{2.9}$$

A δ-name of a point $x \in X$ is a list of all names of all of its basic neighborhoods, while a δ^--name is a list of base elements exhausting the complement of the singleton $\{x\}$. Therefore, δ is the "inner representation" supplying positive information and δ^- is the "outer representation" supplying negative information. A θ-name of an open set W is a list of base elements exhausting W, while a θ^--name is a list of all names of all basic sets intersecting its complement (which is closed in X). Thus, θ is the "inner representation" supplying positive information and θ^- is the "outer representation" supplying negative information. For the closed sets, ψ (the complement of θ^-) is the "inner representation" and ψ^- (the complement of θ) is the "outer representation". Finally, for the multi-representation κ, $K \in \kappa(q)$ if and only if q is a list of all names of all finite unions of base elements that cover K. κ is the "cover representation" of the compact sets. Also, it can be proven [WG09, Lemma 6] that all the inner and outer representations are well defined and that in general, the positive information cannot be obtained from the negative information and vice-versa, that is, $\delta \not\leq_t \delta^- \wedge \delta^- \not\leq_t \delta$; $\theta \not\leq_t \theta^- \wedge \theta^- \not\leq_t \theta$ and $\psi \not\leq_t \psi^- \wedge \psi^- \not\leq_t \psi$ [WG09, Theorem 7]. Notice that the names in Definitions 2.1.6 and 2.1.7 must not be "polluted" by words $w \notin \operatorname{dom}\nu$ since implicitly $w \in \operatorname{dom}\nu$ if $w \ll p \in \operatorname{dom}(\delta)$, $w \in \operatorname{dom}\nu$ if $w \ll p \in \operatorname{dom}(\psi)$ and $w \in \operatorname{dom}(\bigcup_{\text{fin}}\nu) = \operatorname{dom}(\nu^{\text{fin}})$ if $w \ll p \in \operatorname{dom}(\kappa)$. If $\operatorname{dom}\nu$ is computable, these conditions can be checked easily.

Example 2.1.8. Let $\mathbf{R}^n$ be computable euclidean space from Example 2.1.2. For any point $x \in \mathbb{R}^n$, $\delta(p) = x$ if and only if p represents a sequence of all open rational balls $(\mu^n(w))_{w \ll p}$ such that $x \in \mu^n(w)$ for all $w \ll p$. $\psi(q) = \mathbb{S}^{n-1}$ if and only if for all $w \ll q$, $\mu^n(w) \cap \mathbb{S}^{n-1} \neq \varnothing$, that is, q encodes a sequence of all open rational balls that intersect $\mathbb{S}^{n-1}$. As in example 2.1.2, we fix the following convention for all the representations induced by the space $\mathbf{R}^n$: Each representation $\delta, \theta, \psi, \widetilde{\psi}, \kappa$ and $\delta^-, \theta^-, \psi^-$ from Definitions 2.1.6 and 2.1.7 respective, will be denoted as $\delta^n, \theta^n, \psi^n, \widetilde{\psi}^n, \kappa^n$ and $(\delta^-)^n, (\theta^-)^n, (\psi^-)^n$.

Remember from Definition 1.3.25 that given a multi-representation $\gamma\colon \subseteq \Sigma^\omega \to M$ of a set M and $x \in M$, x is γ-computable if and only if $x \in \gamma(p)$ for some computable $p \in \operatorname{dom}\gamma$.

Lemma 2.1.9. *For any $n \geqslant 1$, the closed set $\mathbb{R}^n_+$ is ψ^n,$(\psi^-)^n$-computable in $\mathbf{R}^n$.*

Proof. To see that $\mathbb{R}^n_+$ is ψ^n-computable, a Type-2 machine needs to output all strings $w \in \operatorname{dom}(\mu^n)$ such that
$$\mu^n(w) \cap \mathbb{R}^n_+ \neq \varnothing. \tag{2.10}$$
For each w, this can be checked by a Turing machine in finite time using Theorem 1.2.6, therefore the set $\mathbb{R}^n_+$ is ψ^n-computable. By definition, $\mathbb{R}^n_+$ is $(\psi^-)^n$-computable if anf only if the open set $\mathbb{R}^n - \mathbb{R}^n_+$ is θ^n-computable, which happens if and only if for all $w \in \operatorname{dom}(\mu^n)$, the expression $\mu^n(w) \subset \mathbb{R}^n - \mathbb{R}^n_+$ can be proven algorithmically. This last expression is the negation of (2.10), so that the complement of $\mathbb{R}^n_+$ is θ^n-computable, thus $\mathbb{R}^n_+$ is $(\psi^-)^n$-computable. □

Using similar techniques, we can show that $\partial\mathbb{R}^n_+$ is also ψ^n,$(\psi^-)^n$-computable and $\operatorname{Int}\mathbb{R}^n_+$ is $\theta^n, (\theta^-)^n$-computable in $\mathbf{R}^n$.

Remark. It is not hard to prove that the inner representation δ^n of $\mathbb{R}^n$ is equivalent to the Cauchy representation ρ^n of $\mathbb{R}^n$ (the generalization of the representation defined in equation (1.7)). However, if $n = 1$, δ^1 is not equivalent to the inner representation $\delta_<$ [BHW08] induced by the computable topological space $\mathbf{R}_<$ of Example 2.1.4.

Lemma 2.1.10. *In the computable real line* $\mathbf{R}$*, the irrational numbers* $\pi, e \in \mathbb{R}$ *are* δ^1*-computable points.*

Proof. We show that e is δ^1-computable, the case of π is very similar. We know that we can write the number e as

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}, \tag{2.11}$$

we claim that using this infinite series, we can find for each $m \geqslant 4$, a rational $e_m \in \mathbb{Q}$, such that $e \in B(e_m, 2^{-m})$ (i.e., $d(e, e_m) < 2^{-m}$, this is called a 2^{-m}*-approximation of* e). To do this, it is enough to sum up to $m+1$ terms of the series in Equation (2.11). Indeed, assuming that $m \geqslant 4$, we have that

$$d(e, \textstyle\sum_{n=0}^{m+1} \frac{1}{n!}) = |e - \displaystyle\sum_{n=0}^{m+1} \frac{1}{n!}| \leqslant |\sum_{n=m+1}^{\infty} \frac{1}{n!}| \leqslant \sum_{n=m+1}^{\infty} \frac{1}{n!} < \sum_{n=m+1}^{\infty} \frac{1}{2^{n+1}} = 2^{-(m+1)} < 2^{-m}.$$

So that we can take e_m as $e_m = \sum_{n=0}^{m+1} \frac{1}{n!}$. Let $\mathfrak{M}_e$ be a Type-2 machine with the following program:

1. For each pair $(w, m) \in \mathrm{dom}(\mu^1) \times \mathbb{N}$, execute the following steps:
   1.1 Compute e_m.
   1.2 If $B(e_m, 2^{-m}) \subset \mu^1(w)$, then write $\iota(w)$ on the output tape.
   1.3 Continue with the next pair.

Intuitively, for each pair (w, m), $\mathfrak{M}_e$ computes a 2^{-m}-approximation of e and then it checks if $B(e_m, 2^{-m})$ is inside the open interval represented by the string $w \in \mathrm{dom}(\mu^1)$, if so, $\mathfrak{M}_e$ prints the string $\iota(w)$ on its output tape, thus $\mathfrak{M}_e$ computes a infinite string $p_e \in \Sigma^\omega$.

We now show that $\delta^1(p_e) = e$. First notice that $p_e \in \mathrm{dom}(\delta^1)$, because for every $w \ll p_e$, w was output by $\mathfrak{M}_e$ in Line 1.2, and every string output by $\mathfrak{M}_e$ is the first element of a pair from the set $\mathrm{dom}(\mu^1) \times \mathbb{N}$ (Line 1 of $\mathfrak{M}_e$). Thus $w \in \mathrm{dom}(\mu^1)$. Second, every $w \ll p_e$ satisfies the property $e \in \mu^1(w)$, this is true because in Line 1.2, w is written on the output tape only if

$$B(e_m, 2^{-m}) \subset \mu^1(w). \tag{2.12}$$

But we know that $d(e_m, e) < 2^{-m}$, thus $e \in B(e_m, 2^{-m})$, so that $e \in \mu^1(w)$. Notice also that p_e contains all $w \in \mathrm{dom}(\mu^1)$ such that $e \in \mu^1(w)$, because for every such string w, there exists an $m \in \mathbb{N}$ which satisfies Equation (2.12). We have proven that

$$(\forall w \in \Sigma^*)(w \ll p_e \Leftrightarrow w \in \mathrm{dom}(\mu^1) \text{ and } e \in \mu^1(w)).$$

By Definition 2.1.6, $\delta^1(p_e) = e$. □

Definition 2.1.11 ([KW85, Wei00, Sch02, Sch03])**.** A representation $\gamma\colon \Sigma^\omega \to X$ of a topological space (X, τ) is called *admissible* (with respect to τ) if it is continuous and $\gamma' \leq_t \gamma$ for every continuous function $\gamma'\colon \Sigma^\omega \to X$.

Proposition 2.1.12 ([Wei00])**.** *If* $\mathbf{X} = (X, \tau, \beta, \nu)$ *is an effective topological space, then the representation* δ *is admissible with respect to the topology* τ*.*

It can be proven that all the other (single-valued) representations of Definitions 2.1.6 and 2.1.7 are admissible with respect to appropriated topologies [Sch03]. We now present a result which gives some nice properties of the representations of a computable topological space **X** and the union and intersection operation between subsets of X, the proof can be found in [WG09].

Theorem 2.1.13 ([WG09], Theorem 11). *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable topological space.*

1. *Finite intersection on open sets is (ν^{fin}, θ)-computable and (θ^{fin}, θ)-computable.*
2. *Union on open sets is (θ^{cs}, θ)-computable.*
3. *On closed sets, finite union is $((\psi^{-})^{fin}, \psi^{-})$-computable and intersection is $((\psi^{-})^{cs}, \psi^{-})$-computable.*
4. *On the at most countable collection $\mathcal{B}$ of closed sets, the function $\mathcal{B} \mapsto \overline{\bigcup \mathcal{B}}$ is (ψ^{cs}, ψ)-computable.*
5. *On the compact sets, finite union is (κ^{fin}, κ)-computable.*
6. *The function $(K, A) \mapsto K \cap A$ for compact K and closed A is $(\kappa, \psi^{-}, \kappa)$-computable.*

The following result tell us something about the computability of some basic decision problems in a computable topological space and the representations given in Definition 2.1.6. The result is also proven in [WG09] and will be useful in this thesis.

Lemma 2.1.14 ([WG09], Corollary 14). *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable topological space. Then for all points $x \in X$, open sets W, closed sets A and compact sets K of X,*

1. *"$x \in W$" is (δ, θ)-c.e.*
2. *"$K \subseteq W$" is (κ, θ)-c.e.*
3. *"$A \cap W \neq \varnothing$" is (ψ, θ)-c.e.*
4. *"$K \cap A = \varnothing$" is (κ, ψ^{-})-c.e.*

2.1.2 Predicate spaces

Predicate spaces are used to build an important class of computable topological spaces from very simple assumptions. In particular, we will show in Chapter 3, how to construct the concept of computable manifold using predicate spaces.

Let X be any set and $\sigma \subseteq \mathcal{P}(X)$. We may say "$x$ has property U" if $x \in U$. For each $x \in X$, let

$$\mathcal{P}_x(X) = \{U \in \sigma \mid x \in U\},$$

Definition 2.1.15. Let X be any set.

1. An *effective predicate space* is a triple $\mathbf{Z} = (X, \sigma, \lambda)$ such that $\sigma \subseteq \mathcal{P}(X)$ is countable and $X = \bigcup_{U \in \sigma} U$, $\lambda\colon \subseteq \Sigma^* \to \sigma$ is a notation of σ and the following assertion holds

$$(\forall x, y \in X)(x = y \iff \mathcal{P}_x(X) = \mathcal{P}_y(X)). \tag{2.13}$$

   Z is a *computable predicate space* if dom λ is computable.

2. Define the representation $\delta_{\mathbf{Z}} \colon \subseteq \Sigma^\omega \to X$ of X by

$$\delta_{\mathbf{Z}}(p) = x \iff (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\lambda \text{ and } x \in \lambda(w)).$$

3. Let $T(\mathbf{Z}) = (X, \tau_\lambda, \beta_\lambda, \nu_\lambda)$ where β_λ is the set of all finite intersections of sets from σ, $\nu_\lambda \overset{\text{def}}{=} \bigcap_{\text{fin}} \lambda \colon \subseteq \Sigma^* \to \beta_\lambda$ and τ_λ is the set of all unions of subsets from β_λ.

Each element $U \in \sigma$ is called an *atomic predicate*. By definition, β_λ is the base generated by σ and τ_λ is the topology in the base β_λ. The notation $\nu_\lambda(\iota(u_1)\cdots\iota(u_k)) = \lambda(u_1) \cap \cdots \cap \lambda(u_k)$ can be called the notation by formal finite intersection.

Lemma 2.1.16 ([WG09]). *Let $\mathbf{Z} = (X, \sigma, \lambda)$ be an effective predicate space.*

1. *$T(\mathbf{Z})$ is an effective topological space, which is computable if $\mathbf{Z}$ is computable (that is, if $\operatorname{dom}(\lambda)$ is computable).*

2. *Let δ_λ be the inner representation of points for $T(\mathbf{Z})$. Then $\delta_\lambda \sim \delta_{\mathbf{Z}}$.*

3. *For every representation γ_0 of a subset $Y \subseteq X$, the set $\{(x, U) \in Y \times \sigma \mid x \in U\}$ is (γ_0, λ)-c.e. if and only if $\{(x, V) \in Y \times \beta_\lambda \mid x \in V\}$ is (γ_0, ν_λ)-c.e.*

Proof. 1. Clearly, β_λ is a base of the topology τ_λ on X and ν_λ is a notation of β_λ that has computable domain if λ has computable domain. If $x \neq y$, then by (2.13) there is some $U \in \sigma$ such that $(x \in U \wedge y \notin U)$ or $(x \notin U \wedge y \in U)$. Since $\sigma \subseteq \tau_\lambda$, (X, τ_λ) is a T_0-space. Condition (2.1) holds for $S \overset{\text{def}}{=} \{(u, v, uv) \mid u, v \in \operatorname{dom}\nu_\lambda\}$.

2. By Definition 1.3.27, to prove that $\delta_{\mathbf{Z}} \sim \delta_\lambda$, we need to show that $\delta_{\mathbf{Z}} \leq \delta_\lambda$ and $\delta_\lambda \leq \delta_{\mathbf{Z}}$. We begin with the first case. There is a machine $\mathfrak{M}$ that on input $p \in \Sigma^\omega$ does the following

```
1. For all $k \in \mathbb{N}$ and all subsets $\{v_1, \dots, v_k\}$ such that $v_i \ll p$ for all $i \leqslant k$:
    1.1 write $\iota(\iota(v_1)\cdots\iota(v_k))$ on the output tape
```

In words, $\mathfrak{M}$ computes an infinite string q such that every $z \ll q$ has the form

$$z = \iota(v_1)\cdots\iota(v_k) \quad (k \geqslant 1 \text{ and } v_i \ll p).$$

When $p \in \operatorname{dom}\delta_{\mathbf{Z}}$, $\delta_{\mathbf{Z}}(p) = x$ for some $x \in X$ and each $v_i \ll p$ has the property $v_i \in \operatorname{dom}\lambda$ and $x \in \lambda(v_i)$. Using these facts, we can see that for the string q computed by $\mathfrak{M}$, each $z \ll q$ represents a base element $\lambda(v_1) \cap \cdots \cap \lambda(v_k) \in \beta_\lambda$ of (X, τ_λ) $(k \geqslant 1)$ and because for each i, $x \in \lambda(v_i)$, $x \in \lambda(v_1) \cap \cdots \cap \lambda(v_k)$. Because of the format in which $\mathfrak{M}$ writes the string q, it is immediate that $z \in \operatorname{dom}\nu_\lambda$. All these facts together imply that for the string q,

$$(\forall z \in \Sigma^*)(z \ll q \Leftrightarrow z \in \operatorname{dom}\nu_\lambda \text{ and } x \in \nu_\lambda(z)).$$

By Definition 2.1.6, $\delta_\lambda(q) = x$. Therefore, the function $f_{\mathfrak{M}}$ computed by $\mathfrak{M}$ translates $\delta_{\mathbf{Z}}$-names into δ_λ-names. We conclude that $\delta_{\mathbf{Z}} \leq \delta_\lambda$.

To see that $\delta_\lambda \leq \delta_{\mathbf{Z}}$, we first construct a Type-2 machine $\mathfrak{N}$ with the following program. On the input $q \in \Sigma^\omega$:

```
1. For all $v \ll q$:
   1.1 For each $u \ll v$
      1.1.1 Write $\iota(u)$ on the output tape.
```

Basically, $\mathfrak{N}$ takes the string q and extracts each string $v \ll q$. Then $\mathfrak{N}$ extracts all string $u \ll v$ and outputs all strings of the form $\iota(u)$. If the input string q lies in the domain of δ_λ, then $\delta_\lambda(q) = y$ for $y \in X$ and $\mathfrak{N}$ will take all the strings $u \in \Sigma^*$ such that

$$u \ll v \ll q \quad (v \in \operatorname{dom} \nu_\lambda),$$

and by the definition of v, we have that each $u \ll v$ is such that $u \in \operatorname{dom} \lambda$ and $y \in \lambda(u)$. In summary, the output string of $\mathfrak{N}$, say p, satisfies the condition

$$u \ll p \Leftrightarrow u \in \operatorname{dom} \lambda \text{ and } y \in \lambda(u).$$

Therefore $p \in \operatorname{dom} \delta_{\mathbf{Z}}$ and $\delta_{\mathbf{Z}}(p) = y$, so that the function $f_{\mathfrak{N}}$ computed by $\mathfrak{N}$ translates δ_λ-names into $\delta_{\mathbf{Z}}$-names, that is, $\delta_\lambda \leq \delta_{\mathbf{Z}}$. We have proven that $\delta_{\mathbf{Z}} \leq \delta_\lambda$ and $\delta_\lambda \leq \delta_{\mathbf{Z}}$, by Definition 1.3.27, $\delta_{\mathbf{Z}} \sim \delta_\lambda$.

3. Let $\gamma_0 :\subseteq \Sigma^\omega \to Y$, where $Y \subseteq X$, $E_\sigma = \{(x, U) \in Y \times \sigma \mid x \in U\}$ and $E_{\beta_\lambda} = \{(x, V) \in Y \times \beta_\lambda \mid x \in V\}$. ($\Rightarrow$) Suppose that the set E_σ is (γ_0, λ)-c.e., by Definition 1.3.25 (and the comment after this definition), there is a machine $\mathfrak{M}$ which halts on input (p, u) if and only if $\gamma_0(p) \in \lambda(u)$. Using the machine $\mathfrak{M}$, we build a machine $\mathfrak{N}$ which halts on input (q, v) if and only if $(\gamma_0(q), \nu_\lambda(v)) \in E_{\beta_\lambda}$ and this will imply that the set E_{β_λ} is (γ_0, ν_λ)-c.e. On the input (q, v), the machine $\mathfrak{N}$ executes the following code:

```
1. For each $u \ll v$,
   1.1 Execute $\mathfrak{M}$ with the input $(q, u)$.
2. Halt.
```

We now show that $\mathfrak{N}$ halts on input (q, v) if and only if $(\gamma_0(q), \nu_\lambda(v)) \in E_{\beta_\lambda}$. Assume that $\mathfrak{N}$ halts on (q, v). Then, in Line 1.1, for each string $u \ll v$, $\mathfrak{M}$ halted on the input (q, u), by hypothesis, this happens if and only if $(\gamma_0(q), \lambda(u)) \in E_\sigma$. This means that $q \in \operatorname{dom} \gamma_0$ and that $u \in \operatorname{dom} \lambda$. Also, this implies that

$$\gamma_0(q) \in \bigcap_{u \ll v} \lambda(u),$$

so that $v \in \operatorname{dom} \nu_\lambda$ and $\gamma_0(q) \in \nu_\lambda(v)$. Therefore we have that $(\gamma_0(q), \nu_\lambda(v)) \in E_{\beta_\lambda}$.

Now suppose that $(\gamma_0(q), \nu_\lambda(v)) \in E_{\beta_\lambda}$. Then $v = \iota(u_1) \cdots \iota(u_s)$, where $u_i \in \operatorname{dom} \lambda$ for some $s \geqslant 1$. As $\gamma_0(q) \in \nu_\lambda(v) = \lambda(u_1) \cap \cdots \cap \lambda(u_s)$, it holds for all $i = 1, \ldots, s$, that $\gamma_0(q) \in \lambda(u_i)$, so that the machine $\mathfrak{M}$ will halt on all the input tuples (q, u_i). This says that if the machine $\mathfrak{N}$ is given the input (q, v), it will reach Line 2 in finite time, thus $\mathfrak{N}$ will halt on (q, v). This proves that the machine $\mathfrak{N}$ is correct and that E_{β_λ} is (γ_0, ν_λ)-c.e.

($\Leftarrow$) The argument to show that E_σ is (γ_0, λ)-c.e. if E_{β_λ} is (γ_0, ν_λ)-c.e. is very similar to the previous case, thus we omit it.

□

Roughly speaking, a $\delta_{\mathbf{Z}}$-name of a point is a list of all of its atomic predicates, while a δ_λ-name is a list of all finite intersections of such sets. Clearly, the two representations are equivalent.

Example 2.1.17. Define $\mathbf{Z} \overset{\text{def}}{=} (\mathbb{R}, \sigma, \lambda)$ such that $\operatorname{dom} \lambda = \operatorname{dom} \nu_{\mathbb{Q}}$, $\lambda(w) \overset{\text{def}}{=} (\nu_{\mathbb{Q}}(w), \nu_{\mathbb{Q}}(w)+1) \subset \mathbb{R}$ and $\sigma \overset{\text{def}}{=} \operatorname{range}(\lambda)$. For each $w \in \operatorname{dom} \lambda$, $\lambda(w)$ is the open interval in $\mathbb{R}$ with the endpoints $\nu_{\mathbb{Q}}(w)$ and $\nu_{\mathbb{Q}}(w)+1$. We claim that $\mathbf{Z}$ is a computable predicate space. By definition, the set $\operatorname{dom} \lambda$ is a computable subset of Σ^*, because $\operatorname{dom} \nu_{\mathbb{Q}}$ is computable. Now we need to show that property (2.13) is satisfied by $\mathbf{Z}$, to do this, we will prove that for $x, y \in \mathbb{R}$,

$$x \neq y \Rightarrow \mathcal{P}_x(\mathbb{R}) \neq \mathcal{P}_y(\mathbb{R}).$$

So, assume that we have $x, y \in \mathbb{R}$ such that $x \neq y$. Without loss of generality, suppose that $x < y$. Then we can find $q \in \mathbb{Q}$ such that $x < q < y$ and $d(q,y) < \frac{1}{2}$. This implies that $y \in Q$, where $Q = (q, q+1)$, so that $Q \in \mathcal{P}_y(\mathbb{R})$. Also, as $x < q$, we have that $x \notin Q$, thus $Q \notin \mathcal{P}_x(\mathbb{R})$, therefore $\mathcal{P}_x(\mathbb{R}) \neq \mathcal{P}_y(\mathbb{R})$. So that $\mathbf{Z}$ fulfills Definition 2.1.15. By Lemma 2.1.16, $T(\mathbf{Z}) = (\mathbb{R}, \tau^1, \beta_\lambda, \nu_\lambda)$ is a computable topological space. The computable space $T(\mathbf{Z})$ and the computable space $\mathbf{R}$ from Example 2.1.2 are equivalent as we shall see in Definition 2.2.12. This example can be generalized to show that for any $n \in \mathbb{N}$, $\mathbb{R}^n$ has the structure of a computable predicate space $\mathbf{Z}^n$ such that $T(\mathbf{Z}^n)$ is equivalent to $\mathbf{R}^n$.

Subspaces of computable topological spaces

We need to consider restrictions and products of effective topological spaces [WG09]. Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be an effective topological space. For a subspace $B \subseteq X$, define the restriction $B_{\mathbf{X}} = (B, \tau_B, \beta_B, \nu_B)$ of X to B by $\operatorname{dom}(\nu_B) \overset{\text{def}}{=} \operatorname{dom} \nu, \nu_B(w) \overset{\text{def}}{=} \nu(w) \cap B, \beta_B \overset{\text{def}}{=} \operatorname{range}(\nu_B)$ and $\tau_B \overset{\text{def}}{=} \{W \cap B \mid W \in \tau\}$. Let $\delta_B, \theta_B, \ldots, \psi_B^-$ be the representations for $B_{\mathbf{X}}$ from Definitions 2.1.6 and 2.1.7. Remember from Definition 1.2.2 that for a multi-function $f\colon X \rightrightarrows Y$ and $Z \subseteq Y$, the multi-function $f|^Z\colon X \rightrightarrows Z$ is defined by $f|^Z(x) \overset{\text{def}}{=} f(x) \cap Z$ for all $x \in X$. The next result is proven in [WG09, Lemma 26].

Lemma 2.1.18. *$B_{\mathbf{X}}$ is an effective topological space, which is computable if $\mathbf{X}$ is computable. Also, the following properties are satisfied:*

1. $\delta_B = \delta|^B$,
2. $\theta_B(p) = \theta(p) \cap B$ *for all* $p \in \operatorname{dom}(\theta_B) = \operatorname{dom} \theta$,
3. $\psi_B^-(p) = \psi^-(p) \cap B$ *for all* $p \in \operatorname{dom}(\psi_B^-) = \operatorname{dom} \psi^-$,
4. $\psi_B|^{\mathcal{C}} = \psi|^{\mathcal{C}}$ *for* $\mathcal{C} = \{C \subseteq B \mid C$ *closed in* $X\}$,
5. $\kappa_B|^{\mathcal{L}} = \kappa|^{\mathcal{L}}$ *for* $\mathcal{L} = \{K \subseteq B \mid K$ *compact in* $X\}$.

Example 2.1.19. In the same spirit of Example 2.1.2, We define the *Computable half space* $\mathbf{R}^n_+ = (\mathbb{R}^n_+, \tau^n_+, \beta^n_+, \mu^n_+)$ as the computable subspace $(\mathbb{R}^n_+)_{\mathbf{R}^n}$. Each one of the induced representations $\delta^n_{\mathbb{R}^n_+}, \theta^n_{\mathbb{R}^n_+}, \psi^n_{\mathbb{R}^n_+}, \widetilde{\psi}^n_{\mathbb{R}^n_+}, \kappa^n_{\mathbb{R}^n_+}$ and $(\delta^-)^n_{\mathbb{R}^n_+}, (\theta^-)^n_{\mathbb{R}^n_+}, (\psi^-)^n_{\mathbb{R}^n_+}$ from Definitions 2.1.6 and 2.1.7 respectively, will be denoted by $\delta^n_+, \theta^n_+, \psi^n_+, \widetilde{\psi}^n_+, \kappa^n_+$ and $(\delta^-)^n_+, (\theta^-)^n_+, (\psi^-)^n_+$ respectively. Another good example is the computable subspace $(\partial\mathbb{R}^n_+)_{\mathbf{R}^n}$, induced by the boundary of $\mathbb{R}^n_+$. We denote this space by $\partial\mathbf{R}^n_+ = (\partial\mathbb{R}^n_+, \partial\tau^n, \partial\beta^n, \partial\mu^n)$ and the induced representations by $\partial\delta^n, \partial\theta^n, \partial\psi^n, \partial\widetilde{\psi}^n, \partial\kappa^n$ and $\partial(\delta^-)^n, \partial(\theta^-)^n, \partial(\psi^-)^n$. As $\mathbf{R}^n$, the computable subspaces $\mathbf{R}^n_+$ and $\partial\mathbf{R}^n_+$ will be very useful throughout the rest of the thesis.

The product of computable topological spaces

For $i = 1, 2$ let $\mathbf{X}_i = (X_i, \tau_i, \beta_i, \nu_i)$ be effective topological spaces with representations $\delta_i, \theta_i, \ldots, \psi_i^-$ from Definitions 2.1.6 and 2.1.7. To convert the space $X_1 \times X_2$ with the product topology into an effective topological space define the product

$$\overline{\mathbf{X}} = (X_1 \times X_2, \overline{\tau}, \overline{\beta}, \overline{\nu})$$

of $\mathbf{X}_1$ and $\mathbf{X}_2$, where $\operatorname{dom}\overline{\nu} = \{\langle u_1, u_2\rangle \mid (u_1, u_2) \in \operatorname{dom}\nu_1 \times \operatorname{dom}\nu_2\}$, $\overline{\nu}(\langle u_1, u_2\rangle) = \nu_1(u_1) \times \nu_2(u_2)$, $\overline{\beta} = \operatorname{range}(\overline{\nu})$ and $\overline{\tau}$ is the product topology generated by $\overline{\beta}$. The basic properties of $\overline{\mathbf{X}}$ are proven in [WG09] and are given by the following

Lemma 2.1.20. *$\overline{\mathbf{X}}$ is an effective topological space, which is computable if $\mathbf{X}_1$ and $\mathbf{X}_2$ are computable. Let $\overline{\delta}, \overline{\theta}, \ldots, \overline{\psi}^-$ be the representations for $\overline{\mathbf{X}}$ from Definitions 2.1.6 and 2.1.7. Then*

1. *$\overline{\delta} \sim [\delta_1, \delta_2]$*[2]*.*
2. *The function $(x_1, x_2) \mapsto (x_1, x_2)$ is $(\delta_1, \delta_2, \overline{\delta})$-computable and each projection $(x_1, x_2) \mapsto x_i$ is $(\overline{\delta}, \delta_i)$-computable.*
3. *For open sets, the product $(W_1, W_2) \mapsto W_1 \times W_2$ is $(\theta_1, \theta_2, \overline{\theta})$-computable. Furthermore, the product is $(\theta_1^-, \theta_2^-, \overline{\theta}^-)$-computable if the set $Z_i = \{w \in \Sigma^* \mid \nu_i(w) \neq \varnothing\}$ is c.e. for each $i = 1, 2$.*
4. *For open sets, the projection $W_1 \times W_2 \mapsto W_1$ is $(\overline{\theta}, \theta_1)$-computable if the set Z_2 is c.e. The analogue is true if we replace W_1 and Z_2 by W_2 and Z_1 respectively.*
5. *For closed sets, the product $(A_1, A_2) \mapsto A_1 \times A_2$ is $(\psi_1, \psi_2, \overline{\psi})$-computable and $(\psi_1^-, \psi_2^-, \overline{\psi}^-)$-computable.*
6. *For compact sets, the operation $(K_1, K_2) \mapsto K_1 \times K_2$ is $(\kappa_1, \kappa_2, \overline{\kappa})$-computable and the projection $K_1 \times K_2 \mapsto K_i$ is $(\overline{\kappa}, \kappa_i)$-computable ($i = 1, 2$).*

The generalization to finite products[3] is straightforward. Some examples of computable topological spaces which are subspaces or products of other computable topological spaces will be given later.

2.2 Computable functions between computable topological spaces

In order to have a complete and solid theory of computable topology, it is necessary to work with functions between computable topological spaces which preserve computability. These functions are precisely *computable functions*. To define when a function $f\colon X \to Y$ of the topological spaces X and Y is computable, representations and realizations via computable functions $f\colon \Sigma^\omega \to \Sigma^\omega$

[2]For the explicit definition of the representation $[\delta_1, \delta_2]$, see Definition 1.3.31

[3]In this thesis, we do not need to consider computability for infinite products of computable topological spaces [RW13].

will be very important. In this section we introduce several ways to define computable functions among computable topological spaces. An important property of computability of functions is that it implies continuity. This is reminiscent of the basic facts which state that differentiability and piecewise-linearity imply continuity too. We also introduce two types of equivalences of computable topological spaces by using *computable homeomorphisms* and at the end of the section, we study *computable embeddings*.

2.2.1 Multi-representations for continuous functions

In the work of Weihrauch and Grubba [WG09], a number of multi-representation for the set $CP(X,Y)$ of partial continuous functions between the effective topological spaces $\mathbf{X} = (X,\tau,\beta,\nu)$ and $\mathbf{Y} = (Y,\tau',\beta',\nu')$ are introduced. In fact, up to eight different multi-representations for continuous functions are defined and their various equivalences are proven in [WG09]. As we will be dealing with computable functions between computable topological spaces, we need to use some of these multi-representations.

A partial function $f\colon \subseteq X \to Y$ is continuous if and only if for every $W \in \tau'$, $f^{-1}[W]$ is open in $\operatorname{dom} f$, that is, $f^{-1}[W] = V \cap \operatorname{dom} f$ for some $V \in \tau$. The following conditions are equivalent:

C1 f is continuous,

C2 $(\forall x \in \operatorname{dom} f, W \in \tau')(f(x) \in W \Rightarrow (\exists V \in \tau)(x \in V \wedge f[V \cap \operatorname{dom} f] \subseteq W))$,

C3 $f[\mathrm{cls}_{\operatorname{dom} f}(C)] \subseteq \overline{f[C]}$ for every $C \subseteq \operatorname{dom} f$,

The equivalences of **C1**, **C2** and **C3** are well-known [Eng89, Mun00]. Type-2 theory gives us a surprising connection between topology and computability, this is another equivalence for continuity in terms of continuous functions from Σ^ω to itself [Wei00, Theorem 3.2.11].

Theorem 2.2.1. *Let $\mathbf{X} = (X,\tau,\beta,\nu)$ and $\mathbf{Y} = (Y,\tau',\beta',\nu')$ be effective topological spaces. Then a map $f\colon \subseteq X \to Y$ is continuous if and only if f has a continuous (δ,δ')-realization.*

This is the "main theorem" for admissible representations, since for an effective topological space $\mathbf{X} = (X,\tau,\beta,\nu)$, by Proposition 2.1.12, the representation of points δ is admissible with respect to the topology τ. We use these and other characterizations to define a number of multi-representations of the set of partial continuous functions $CP(X,Y)$.

Definition 2.2.2. For the effective topological spaces $\mathbf{X} = (X,\tau,\beta,\nu)$ and $\mathbf{Y} = (Y,\tau',\beta',\nu')$ with inner representations $\delta,\theta,\psi,\kappa$ and $\delta',\theta',\psi',\kappa'$ respectively, define the multi-representations $\overrightarrow{\delta_i} : \subseteq \Sigma^\omega \rightrightarrows CP(X,Y)$ $(i = 1,2,3,4,5)$ as follows[4]:

1. $f \in \overrightarrow{\delta_1}(p) \Longleftrightarrow f \circ \delta(q) = \delta' \circ \eta_p^{\omega\omega}(q)$ for all $q \in \operatorname{dom}(f \circ \delta)$,

2. $f \in \overrightarrow{\delta_2}(p) \Longleftrightarrow f^{-1}[\theta'(q)] = \theta \circ \eta_p^{\omega\omega}(q) \cap \operatorname{dom} f$ for all $q \in \operatorname{dom}(\theta')$,

3. $f \in \overrightarrow{\delta_3}(p) \Longleftrightarrow f^{-1}[\nu'(w)] = \theta \circ \eta_p^{*\omega}(w) \cap \operatorname{dom} f$ for all $w \in \operatorname{dom}(\nu')$,

4. $f \in \overrightarrow{\delta_4}(p) \Longleftrightarrow \overline{f[C]} = \psi' \circ \eta_p^{\omega\omega}(q)$ if $C \subseteq \operatorname{dom} f$ and $\overline{C} = \psi(q)$,

[4]The representations η^{ab} $(a,b \in \{*,\omega\})$ are introduced in Section 1.3.4.

5. $f \in \overrightarrow{\delta_5}(p) \iff f[K] \in \kappa' \circ \eta_p^{\omega\omega}(q)$ if $K \subseteq \operatorname{dom} f$ and $K \in \kappa(q)$.

We remark that in [Wei00, Wei08], $\overrightarrow{\delta_1}$ is denoted by $[\delta \to_p \delta']$, but in this thesis, we will not use this notation. If we call the infinite string p a "program" to compute the function η_p^{ab} on any input, then in part 1. of Definition 2.2.2, a name p of f is a program to compute f with respect to the inner representations δ and δ'; in 2. a name p is a program for computing the function $W \mapsto f^{-1}[W]$ with respect to θ and θ', etc. It is shown in [WG09] that if $\mathbf{X}, \mathbf{Y}$ are computable, then

$$\overrightarrow{\delta_1} \sim \overrightarrow{\delta_2} \sim \overrightarrow{\delta_3} \sim \overrightarrow{\delta_4} \sim \overrightarrow{\delta_5}, \tag{2.14}$$

and that each class $\overrightarrow{\delta_i}$ is closed under restriction. Also, if we restrict each multi-representation $\overrightarrow{\delta_i}$ to a class of continuous functions with fixed domain, they become singled-valued. This is true because of the following result, which is proven in [WG09].

Theorem 2.2.3. *Let $\mathbf{X}_1, \mathbf{X}_2$ be computable topological spaces. Then for every i with $1 \leqslant i \leqslant 5$ and every $p \in \operatorname{dom}(\overrightarrow{\delta_i})$, $f(x) = g(x)$ if $f, g \in \overrightarrow{\delta_i}(p)$ and $x \in \operatorname{dom} f \cap \operatorname{dom} g$.*

Definition 2.2.4. Let $\mathbf{X}$ and $\mathbf{Y}$ be as in Definition 2.2.2. A (partial) continuous function $f\colon \subseteq X \to Y$ is *computable* if there exists a Type-2 machine $\mathfrak{M}_f$, such that $\mathfrak{M}_f$ can compute an element $p \in \operatorname{dom}(\overrightarrow{\delta_i})$ for some $i \in \{1, 2, 3, 4, 5\}$.

Notice that if $p \in \operatorname{dom}(\overrightarrow{\delta_i})$ is computable, then by Lemma 1.3.35, the function η_p^{ab} is computable ($a, b \in \{*, \omega\}$), so that combining this fact with Definition 2.2.2 and Equation (2.14), we deduce that f is a computable function between the spaces $\mathbf{X}$ and $\mathbf{Y}$ if and only if one of the following equivalent conditions hold:

- f is (δ, δ')-computable.
- The function $W \mapsto f^{-1}[W]$ is (θ', θ)-computable.
- The function $B \mapsto f^{-1}[B]$ is (ν', θ)-computable.
- The function $\overline{C} \mapsto \overline{f[C]}$ is (ψ, ψ')-computable.
- The function $K \mapsto f[K]$ is (κ, κ')-computable.

We present the following results as examples of computable functions. Let $\overline{\mathbf{R}} = (\mathbb{R} \times \mathbb{R}, \overline{\tau}, \overline{\beta}, \overline{\nu})$, where the computable topology of $\overline{\mathbf{R}}$ is induced by the computable real line $\mathbf{R}$.

Lemma 2.2.5. *In the computable topological spaces $\mathbf{R}$ and $\overline{\mathbf{R}}$, the following statements hold.*

(a) The sum $(x, y) \mapsto x + y$ and multiplication $(x, y) \mapsto xy$ are computable.

(b) The square root function $\sqrt{\ }\colon \subseteq \mathbb{R} \to \mathbb{R}$ is computable.

Proof. (a) We show that sum is a computable function from $\overline{\mathbf{R}}$ to $\mathbf{R}$, the case of multiplication follows from a similar argument. We first prove that there exists a Type-2 machine $\mathfrak{M}$ such that given any $p \in \operatorname{dom}(\delta^1)$ and $m \in \mathbb{N}$, $\mathfrak{M}$ can output a rational $q \in \mathbb{Q}$ that satisfies the equation

$$\delta^1(p) \in B(q, 2^{-m}). \tag{2.15}$$

Let $\mathfrak{M}$ be the machine with the following program. On input (p, m):

1. For each $u \ll p$,

    1.1 Let $\mu^1(u)$ be the ball $B_u = B(q,r)$. If $B_u \subset B(q,2^{-m})$ then write q on the output tape and halt.

    1.2 Otherwise, continue with the next iteration.

Notice that each step of $\mathfrak{M}$ can be computed in finite time, as the if's test can be done using Theorem 1.2.6. Now we need to show that if the input pair lies in the set $\operatorname{dom}(\delta^1) \times \mathbb{N}$, $\mathfrak{M}$ always halts and the output value satisfies Equation (2.15). Suppose that $(p,m) \in \operatorname{dom}(\delta^1) \times \mathbb{N}$. Then p is a list of all the intervals in $\mathbb{R}$ which contain the real number $x = \delta^1(p)$. Thus there must exists $u \ll p$ such that $\mu^1(u) = B(q,r) \subset B(q,2^{-m})$. Therefore $\mathfrak{M}$ will halt in finite time with the rational q written on its output tape. As $x \in B(q,r)$, it follows that $\delta^1(p) = x \in B(q,2^{-m})$, so that $\mathfrak{M}$ is correct and always halts if the input lies in the set $\operatorname{dom}(\delta^1) \times \mathbb{N}$.

We are ready to show that the sum function $+\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is computable. Let $\mathfrak{M}_+$ be a machine which on input $s \in \operatorname{dom}\overline{\delta}$ ($\overline{\delta}(s) = (x,y)$) does the following:

1. For all $(v,m) \in \operatorname{dom}(\mu^1) \times \mathbb{N}$, execute the following steps

    1.1 Compute $q_+ \in \mathbb{Q}$ such that $d(q_+, x+y) < 2^{-m}$.

    1.2 If $B(q_+, 2^{-m}) \subset \mu^1(v)$, then write $\iota(v)$ on the output tape.

    1.3 Continue with the next pair.

The argument to show that $\mathfrak{M}_+$ is correct is almost the same argument that we used in Lemma 2.1.10, except for Step 1.1 of $\mathfrak{M}_+$. To prove that this step can be carried on in finite time by $\mathfrak{M}_+$, we only need to show that from the input string $s \in \operatorname{dom}\overline{\delta}$, we can compute the rational q_+, which is a 2^{-m}-approximation of $x+y$. as $\overline{\delta}(s) = (x,y) \in \mathbb{R} \times \mathbb{R}$ and $\overline{\mathbf{R}}$ is a computable product space, we can use the string s and apply part 2 of Lemma 2.1.20 to compute strings $p_x, p_y \in \operatorname{dom}\delta^1$ such that $\delta^1(p_x) = x$ and $\delta^1(y) = y$. We can then use the machine $\mathfrak{M}$ on the input pairs $(p_x, m+1), (p_y, m+1)$ to obtain two rationals $q_x, q_y \in \mathbb{Q}$ such that q_x and q_y are $2^{-(m+1)}$-approximations of x and y respectively. We claim that $q_x + q_y$ is a 2^{-m}-approximation of $x+y$. Indeed,

$$d(x+y, q_x+q_y) = \left|(x+y) - (q_x+q_y)\right| \leqslant \left|x - q_x\right| + \left|y - q_y\right| < 2 \cdot 2^{-(m+1)} = 2^{-m},$$

thus $q_+ = q_x + q_y$ is a 2^{-m}-approximation of $x+y$. Therefore, step 1.1 of $\mathfrak{M}_+$ can be executed in finite time and it depends only on the input string $s \in \operatorname{dom}\overline{\delta}$. So that the function $F_+\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ computed by $\mathfrak{M}_+$, is a computable realization of the sum function. We conclude that sum is $(\overline{\delta}, \delta^1)$-computable, by Definition 2.2.4, sum is computable.

(b) The argument to show that the square root is computable, is very similar to the previous case. Using the machine $\mathfrak{M}$ to compute 2^{-m}-approximations of $x \in \mathbb{R}$, we can find 2^{-m}-approximations of $\sqrt{x}$ by using the expression

$$\sqrt{x} = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(1-2n)(n!)^2 4^n} (x-1)^n.$$

We omit the details. □

More examples of computable functions exist. Almost all known real functions are computable, like the trigonometric functions and their inverses, the exponential and logarithm functions (with computable bases) [Wei00, Bra05]. We will present many more examples in Chapter 3.

Lemma 2.2.6. *Let* $\mathbf{X}, \mathbf{Y}$ *and* $\mathbf{Z}$ *be computable topological spaces. If* $f\colon \subseteq X \to Y$ *and* $g\colon \subseteq Y \to Z$ *are computable maps such that* $\operatorname{range}(f) \cap \operatorname{dom}(g) \neq \varnothing$*, then* $g \circ f\colon \subseteq X \to Z$ *is a computable map from* $\mathbf{X}$ *to* $\mathbf{Z}$*.*

Proof. Let δ, δ' and δ'' be the inner representations for X, Y and Z respectively. If f, g are computable maps, then by Definition 2.2.4, f is (δ, δ')-computable and g is (δ', δ'')-computable. This means that there exist $F, G\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ such that F is a computable realization of f and G is a computable realization of g, that is,

$$f \circ \delta = \delta' \circ F \quad \text{and} \quad g \circ \delta' = \delta'' \circ G.$$

The composition $G \circ F \subseteq \Sigma^\omega \to \Sigma^\omega$ is computable [Wei00, Wei08, TW11], thus for any $p \in \operatorname{dom}(\delta)$ such that $f(\delta(p)) \in \operatorname{dom} g$,

$$(g \circ f) \circ \delta(p) = g \circ (f \circ \delta)(p) = g \circ (\delta' \circ F)(p) = (g \circ \delta') \circ F(p) = (\delta'' \circ G) \circ F(p) = \delta'' \circ (G \circ F)(p),$$

and we conclude that $(g \circ f) \circ \delta = \delta'' \circ (G \circ F)$, so that $G \circ F$ is a computable realization of $g \circ f$, that is, it is (δ, δ'')-computable, by Definition 2.2.4, $g \circ f$ is a computable function between $\mathbf{X}$ and $\mathbf{Z}$. This concludes the proof. $\square$

2.2.2 Equivalences between computable topological spaces

We will be using two types of equivalences between computable topological spaces. The first one is that of *computable homeomorphism*.

Definition 2.2.7. Let $\mathbf{X} = (X, \tau, \beta, \nu)$ and $\mathbf{X}' = (X', \tau', \beta', \nu')$ be computable topological spaces . A *computable homeomorphism* is a map $h\colon X \to X'$ such that h is a homeomorphism and also h and h^{-1} are computable functions . When such a map h exists, we say that $\mathbf{X}$ and $\mathbf{X}'$ are *computably homeomorphic* . This fact is denoted by $\mathbf{X} \cong_{ct} \mathbf{X}'$.

Sometimes, if there is no confusion about which (computable) topologies we are considering in the sets X, X', we will just say that X and X' themselves are computably homeomorphic.

Example 2.2.8. Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be any computable topological space. Then $\mathbf{X} \cong_{ct} \mathbf{X}$ because the identity map $1_X\colon X \to X$ is clearly a computable homeomorphism.

Example 2.2.9. Let $B = B(0,1) \subset \mathbb{R}^n$ be the open unit ball with the (computable) subspace topology (see Lemma 2.1.18). The function $h\colon B \to \mathbb{R}^n$ defined by $h(x) = \frac{x}{1-\|x\|^2}$ is a (δ_B, δ^n)-computable map and its inverse is the function $h^{-1}\colon \mathbb{R}^n \to B$, given by

$$h^{-1}(y) = \frac{2y}{1 + \sqrt{1 + 4\|y\|^2}},$$

which is (δ^n, δ_B)-computable. According to Definition 2.2.4, h is a computable homeomorphism between $B_{\mathbf{R}^n}$ and $\mathbf{R}^n$.

Example 2.2.10. Let $\mathbf{S}^n = \mathbb{S}^n_{\mathbf{R}^{n+1}}$ denote the n-sphere $\mathbb{S}^n$ equipped with the computable subspace topology induced by $\mathbf{R}^{n+1}$. An important example of a computable homeomorphism is the stereographic projection $s\colon S_P \to \mathbb{R}^n$ ($S_P = \mathbb{S}^n - \{P\}, P = (0,\dots,0,1)$) onto euclidean space $\mathbb{R}^n$ given by the equation

$$s(x,t) = \frac{x}{1-t} \text{ with } x = (x_1,\dots,x_n).$$

Because s is computable, we have that $(S_P)_{\mathbf{S}^n} \cong_{ct} \mathbf{R}^n$. The homeomorphism s has been fundamental for the work in [Zho96].

The following result about half spaces will be needed later.

Lemma 2.2.11. *Let $n,m \geqslant 1$, $X = \mathbb{R}^n_+ \times \mathbb{R}^m_+$ and $\overline{\mathbf{X}} = (X,\overline{\tau},\overline{\beta},\overline{\nu})$ be the computable topological space induced by the product X. Then the computable topological spaces $\overline{\mathbf{X}}$ and $\mathbf{R}^{n+m}_+$ are computably homeomorphic.*

Proof. Throughout this proof, all the topological product spaces have the obvious computable topologies. First notice that for any $n \geqslant 1$, $\mathbb{R}^n_+$ and $\mathbb{R}^{n-1} \times \mathbb{R}_+$ are computably homeomorphic by means of the map $(x_1,\dots,x_n) \mapsto ((x_1,\dots,x_{n-1}),x_n)$. Second, for any $n,m \geqslant 1$,

$$\mathbb{R}^n_+ \times \mathbb{R}^m_+ \cong_{ct} \mathbb{R}^{n-1} \times \mathbb{R}_+ \times \mathbb{R}^{m-1} \times \mathbb{R}_+ \cong_{ct} \mathbb{R}^{n+m-2} \times \mathbb{R}_+ \times \mathbb{R}_+.$$

Therefore, to complete the proof, we only need to show that $\mathbb{R}_+ \times \mathbb{R}_+$ is computably homeomorphic to $\mathbb{R} \times \mathbb{R}_+$. Let $Q = \mathbb{R}_+ \times \mathbb{R}_+$ and $H = \mathbb{R} \times \mathbb{R}_+$. Define $f\colon Q \to H$ and $g\colon H \to Q$ as

$$f(a,b) = (a^2 - b^2, 2ab);$$

$$g(c,d) = \Big(\sqrt{\tfrac{c+\sqrt{c^2+d^2}}{2}}, \sqrt{\tfrac{-c+\sqrt{c^2+d^2}}{2}}\Big).$$

It is routine to check that f,g are continuous and they are inverse to each other, thus f is a homeomorphism of Q onto H. Now we need to prove that f and g are computable, to see that this is true, first notice that the coordinate functions of f and g are given in terms of compositions of sums, multiplications and square roots. So that in order to see that f and g are computable, it is enough to show that the respective coordinate functions are computable. This can be done by using Lemma 2.2.5, which tell us that the functions sum, multiplication and square root are computable and these facts can be combined with Lemma 2.2.6 which shows that function composition is computable, thus the machines that compute f and g are made up of various subprograms which are build by using Lemmas 2.2.5 and 2.2.6, giving the full specification of these machines and proving their correctness is very similar to previous results, thus we omit the details. Therefore $Q \cong_{ct} H$ and finally

$$\mathbb{R}^{n+m-2} \times \mathbb{R}_+ \times \mathbb{R}_+ \cong_{ct} \mathbb{R}^{n+m-2} \times \mathbb{R} \times \mathbb{R}_+ \cong_{ct} \mathbb{R}^{n+m-1} \times \mathbb{R}_+ \cong_{ct} \mathbb{R}^{n+m}_+$$

and the proof is completed. □

Our second equivalence between computable topological spaces is as follows.

Definition 2.2.12 ([WG09])**.** The computable topological spaces $\mathbf{X} = (X,\tau,\beta,\nu)$ and $\mathbf{X}' = (X,\tau,\beta',\nu')$ are *equivalent* if and only if $\nu \leq \theta'$ and $\nu' \leq \theta$.

The equivalence given in the previous definition identifies when the two notations ν, ν' of the bases β and β' respectively, induce the same computability on the space (X, τ). Example 2.1.17 shows two equivalent computable topological spaces. The computability concepts introduced in Definitions 2.1.6 and 2.1.7 can be called "computationally robust", since they are the same for equivalent computable topological spaces. Usually, non-robust concepts [Wei00, BW99, BP03] have only few applications. The next result, which can be found in [WG09, Theorem 22.2] is very useful in practice.

Theorem 2.2.13. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ and $\mathbf{X}' = (X, \tau, \beta', \nu')$ be computable topological spaces. The following statements are equivalent:*

- *$\mathbf{X}$ and $\mathbf{X}'$ are equivalent;*
- *$\delta \sim \delta'$;*
- *$\theta \sim \theta'$*

Moreover, if $\mathbf{X}$ and $\mathbf{X}'$ are equivalent, then $\gamma \sim \gamma'$ for every representation γ from Definitions 2.1.6 and 2.1.7, where γ' is the representation for $\mathbf{X}'$ corresponding to γ.

Corollary 2.2.14. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ and $\mathbf{X}' = (X, \tau, \beta', \nu')$ be computable topological spaces. Then $\mathbf{X}$ and $\mathbf{X}'$ are equivalent if and only if $1_X \colon X \to X$ is a computable homeomorphism.*

Proof. Let δ, δ' denote the inner representation of points of X in $\mathbf{X}$ and $\mathbf{X}'$ respectively. The identity 1_X is a computable homeomorphism if and only if, by Definition 2.2.4, 1_X is (δ, δ')-computable and also (δ', δ)-computable. This happens if and only if $\delta \leq \delta'$ and $\delta' \leq \delta$, i.e., $\delta \sim \delta'$ (Definition 1.3.27), which is equivalent, by Theorem 2.2.13, to $\mathbf{X}$ and $\mathbf{X}'$ being equivalent spaces. □

The induced effective topological space

Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be an effective topological space and Y a set and suppose that there exists a bijective function $f \colon X \to Y$. Because (X, τ) is a topological space, f induces a topology τ_f on Y given by the base

$$\beta_f = \{f[V] \mid V \in \beta\}.$$

We define a notation $\nu_f \colon \subseteq \Sigma^* \to \beta_f$ of the base β_f as $\nu_f(w) = f[\nu(w)]$ ($\operatorname{dom} \nu_f = \operatorname{dom} \nu$). The 4-tuple $(Y, \tau_f, \beta_f, \nu_f)$ will be denoted by $\mathbf{Y}_f$. It is easy to see that f becomes a homeomorphism between the two spaces (X, τ) and (Y, τ_f). The following results tell us that this fact still holds in a computable way.

Lemma 2.2.15. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be an effective topological space and Y a set. If $f \colon X \to Y$ is a bijective function, then*

a) $\mathbf{Y}_f$ is an effective topological space, which is computable if $\mathbf{X}$ is computable. Also, for any $w \in \operatorname{dom} \nu_f, \nu_f(w) = f[\nu(w)]$.

b) f becomes a computable homeomorphism between $\mathbf{X}$ and $\mathbf{Y}_f$.

c) Let γ be any of the representations given in Definitions 2.1.6 and 2.1.7 induced by $\mathbf{X}$ and let γ_f be the respective representation induced by $\mathbf{Y}_f$. Then

1. $\operatorname{dom}\gamma_f = \operatorname{dom}\gamma$.
2. *If* $\gamma \in \{\delta, \delta^-\}$*, then* $\gamma_f = f \circ \gamma$.
3. *If* $\gamma \in \{\theta, \theta^-, \psi, \psi^-\}$*, then* $\gamma_f(p) = f[\gamma(p)]$.
4. *If* $\gamma \in \{\widetilde{\psi}, \kappa\}$*, then* $\gamma_f(p) = \{f[D] \mid D \in \gamma(p)\}$.

Proof. $a)$ It is clear that $\mathbf{Y}_f$ is an effective topological space. If $\mathbf{X}$ is computable, then $\operatorname{dom}\nu$ is computable and there exists a c.e. set $S \subseteq \operatorname{dom}\nu \times \operatorname{dom}\nu \times \operatorname{dom}\nu$ such that equation $\boxed{2.1}$ of Definition 2.1.1 is satisfied. As $\operatorname{dom}\nu_f = \operatorname{dom}\nu$, $\operatorname{dom}\nu_f$ is computable and the same c.e. set S can be used to fulfill $\boxed{2.1}$ for the effective space $\mathbf{Y}_f$ in Definition 2.1.1. Thus $\mathbf{Y}_f$ is a computable topological space.

$b)$ We can show that f is a computable homeomorphism between $\mathbf{X}$ and $\mathbf{Y}_f$ by proving that the functions

$$B \mapsto f^{-1}[B] \quad \text{and} \quad A \mapsto (f^{-1})^{-1}[A], \quad (B \in \beta, A \in \beta_f),$$

are (ν_f, θ)-computable and (ν, θ_f)-computable respectively. The proof is straightforward, thus we omit it.

$c)$ We argue by cases on the induced representations γ and γ_f:

Case $\gamma = \delta$. Let $p \in \Sigma^\omega$, if $\delta(p) = x \in X$, then $f(x) \in Y$ and

$$\begin{aligned}
\delta(p) = x \quad &\Leftrightarrow \quad (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu \wedge x \in \nu(w)) \\
&\Leftrightarrow \quad (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu_f \wedge f(x) \in f[\nu(w)]) \\
&\Leftrightarrow \quad (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu_f \wedge f(x) \in \nu_f(w)) \\
&\Leftrightarrow \quad \delta_f(p) = f(x).
\end{aligned}$$

Thus $\operatorname{dom}\delta = \operatorname{dom}\delta_f$ and also $\delta_f(p) = f(x) = f(\delta(p)) = f \circ \delta(p)$ for any $p \in \operatorname{dom}\delta_f$. Therefore, $\delta_f = f \circ \delta$.

Case $\gamma = \delta^-$. This case is similar to the previous one.

Case $\gamma = \theta$. Given $p \in \Sigma^\omega$, we have that

$$\begin{aligned}
\theta(p) = W \quad &\Leftrightarrow \quad \begin{cases} w \ll p \Rightarrow w \in \operatorname{dom}\nu, \\ W = \bigcup\{\nu(w) \mid w \ll p\}, \end{cases} \\
&\Leftrightarrow \quad \begin{cases} w \ll p \Rightarrow w \in \operatorname{dom}\nu_f, \\ f[W] = \bigcup\{f[\nu(w)] \mid w \ll p\}, \end{cases} \\
&\Leftrightarrow \quad \begin{cases} w \ll p \Rightarrow w \in \operatorname{dom}\nu_f, \\ f[W] = \bigcup\{\nu_f(w) \mid w \ll p\}, \end{cases} \\
&\Leftrightarrow \quad \theta_f(p) = f[W],
\end{aligned}$$

so that $\operatorname{dom}\theta = \operatorname{dom}\theta_f$ and $\theta_f(p) = f[W] = f[\theta(p)]$ for any $p \in \operatorname{dom}\theta_f$.

Case $\gamma = \psi$. For $p \in \Sigma^\omega$,

$$\begin{aligned}
\psi(p) = A &\Leftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu \wedge A \cap \nu(w) \neq \varnothing)\\
&\Leftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu_f \wedge f[A] \cap f[\nu(w)] \neq \varnothing)\\
&\Leftrightarrow (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom}\nu_f \wedge f[A] \cap \nu_f(w) \neq \varnothing)\\
&\Leftrightarrow \psi_f(p) = f[A].
\end{aligned}$$

And we conclude this case with the same arguments we use in all previous cases.

Cases $\gamma = \theta^-, \psi^-$. These cases follow immediately from the definitions of the representations θ^- and ψ^- and the two previous cases.

Case $\gamma = \kappa$. Let $p \in \Sigma^\omega$, notice first that if $\kappa(p)$ is defined, then $\kappa(p)$ is in general a set of compact subsets of X and accordingly to (2.6),

$$K \in \kappa(p) \iff (\forall z \in \Sigma^*)(z \ll p \Leftrightarrow K \subseteq \textstyle\bigcup_{\text{fin}} \nu(z)),$$

and $\bigcup_{\text{fin}} \nu(z) = \nu(w_1) \cup \cdots \cup \nu(w_s)$ with $w_i \in \operatorname{dom}\nu \wedge w_i \ll z$ for all $i = 1, \ldots, s$ (see Definition 1.3.32 and the remark after Lemma 1.3.33). As each $w_i \in \operatorname{dom}\nu_f$, we have that $f[\bigcup_{\text{fin}} \nu(z)] = f[\nu(w_1) \cup \cdots \cup \nu(w_s)] = f[\nu(w_1)] \cup \cdots \cup f[\nu(w_s)] = \nu_f(w_1) \cup \cdots \cup \nu_f(w_s) = \bigcup_{\text{fin}} \nu_f(z)$. Thus for a compact set $K \in \kappa(p)$,

$$\begin{aligned}
K \in \kappa(p) &\Leftrightarrow (\forall z \in \Sigma^*)(z \ll p \Leftrightarrow K \subseteq \textstyle\bigcup_{\text{fin}} \nu(z)),\\
&\Leftrightarrow (\forall z \in \Sigma^*)(z \ll p \Leftrightarrow f[K] \subseteq f[\textstyle\bigcup_{\text{fin}} \nu(z)]),\\
&\Leftrightarrow (\forall z \in \Sigma^*)(z \ll p \Leftrightarrow f[K] \subseteq \textstyle\bigcup_{\text{fin}} \nu_f(z)),\\
&\Leftrightarrow f[K] \in \kappa_f(p).
\end{aligned}$$

This shows that $\operatorname{dom}\kappa = \operatorname{dom}\kappa_f$ and also that $\kappa_f(p) = \{f[K] \mid K \in \kappa(p)\}$.

Case $\gamma = \widetilde{\psi}$. This is similar to the case $\gamma = \kappa$.

This concludes all the cases for the induced representations on $\mathbf{X}$ and $\mathbf{Y}_f$ and finishes the proof. $\square$

This result tell us that the only essential difference between $\mathbf{X}$ and $\mathbf{Y}_f$ is the "abstract symbol" f. But more can be said if Y already possesses a computable topology.

Corollary 2.2.16. *Let $\mathbf{X} = (X, \tau_X, \beta_X, \nu_X)$, $\mathbf{Y} = (Y, \tau_Y, \beta_Y, \nu_Y)$ be computable topological spaces. If $f\colon X \to Y$ is a computable homeomorphism, then $\mathbf{Y}$ and $\mathbf{Y}_f$ are equivalent.*

Proof. Let δ_X, δ_Y be the inner representations induced by $\mathbf{X}$ and $\mathbf{Y}$ respectively. We will show that $\delta_f \sim \delta_Y$. Because f and f^{-1} are computable functions between the computable spaces $\mathbf{X}$ and $\mathbf{Y}$, they are computable with respect to δ_X and δ_Y, thus there exist computable functions $F, G\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ such that the two inner squares in the following diagram

$$\begin{tikzcd}
X \arrow[r, "f"] & Y \arrow[r, "f^{-1}"] & X \\
\Sigma^\omega \arrow[u, "\delta_X"] \arrow[r, "F"] & \Sigma^\omega \arrow[u, "\delta_Y"] \arrow[r, "G"] & \Sigma^\omega \arrow[u, "\delta_X"]
\end{tikzcd} \tag{2.16}$$

commute. By part (c) of Lemma 2.2.15 and the first square, $\delta_f = f \circ \delta_X = \delta_Y \circ F$, that is, $\delta_f \leq \delta_Y$. By the second square, $f^{-1} \circ \delta_Y = \delta_X \circ G$, so that $\delta_Y = f \circ \delta_X \circ G = \delta_f \circ G$, thus $\delta_Y \leq \delta_f$. This proves that $\delta_Y \sim \delta_f$ and then we can apply Theorem 2.2.13 to deduce that $\mathbf{Y}$ and $\mathbf{Y}_f$ are equivalent □

For a computable predicate space $\mathbf{Z} = (X, \sigma, \lambda)$, we have the computable space $T(\mathbf{Z}) = (X, \tau, \beta_\lambda, \nu_\lambda)$ of Lemma 2.1.16, where $\nu_\lambda(\iota(u_1)\cdots\iota(u_k)) = \lambda(u_1) \cap \cdots \cap \lambda(u_k)$, τ is the topology generated by the subbase σ and $\delta_\lambda \sim \delta_{\mathbf{Z}}$. If σ is not only a subbase, but a base of τ, then we can construct the effective space $\mathbf{Y} \overset{\text{def}}{=} (X, \tau, \sigma, \lambda)$, which may be computable. For the topology τ we have on the one hand, the basis β_λ with the notation ν_λ (defined via formal intersection of subbase elements) and on the other hand, the basis σ with notation λ. The question is: Are $T(\mathbf{Z})$ and $\mathbf{Y}$ equivalent ? The answer is right here [WG09, Lemma 23].

Lemma 2.2.17. *Let $\mathbf{Y} = (X, \tau, \sigma, \lambda)$ be an effective topological space such that $\mathbf{Z} = (X, \sigma, \lambda)$ is a computable predicate space. Then $T(\mathbf{Z})$ and $\mathbf{Y}$ are equivalent if and only if $\mathbf{Y}$ is a computable topological space.*

2.2.3 Computable embeddings

Sometimes it is useful to have a space X inside another (probably bigger) space Y, that is, when X is *embedded* in Y. This idea is formalized with the concept of *embedding*. A continuous function $h\colon X \to Y$ is a *(topological) embedding* if h is a homeomorphism of X onto $h[X] \subset Y$, where $h[X]$ is equipped with the subspace topology induced by Y. Such an embedding is denoted by

$$h\colon X \hookrightarrow Y \text{ or } X \overset{h}{\hookrightarrow} Y.$$

In this way, we can view X as a subspace of Y. The corresponding definition for computable topological spaces is that of *computable embedding*.

Definition 2.2.18. Let $\mathbf{X} = (X, \tau_X, \beta_X, \nu_X)$, $\mathbf{Y} = (Y, \tau_Y, \beta_Y, \nu_Y)$ be computable topological spaces. A *computable embedding* of $\mathbf{X}$ into $\mathbf{Y}$ is a topological embedding $h\colon X \hookrightarrow Y$ such that h is a computable homeomorphism of $\mathbf{X}$ onto the computable subspace $X'_{\mathbf{Y}}$, where $X' = h[X]$.

By Corollary 2.2.16, the computable topological spaces $\mathbf{X}'_h = (h[X], \tau_h, \beta_h, \nu_h)$ and $X'_{\mathbf{Y}}$ are equivalent.

Example 2.2.19. For any effective topological space $\mathbf{X} = (X, \tau, \beta, \nu)$ and any $Z \subseteq X$, the computable topological space $Z_{\mathbf{X}}$ is embedded in X via the inclusion map $i\colon Z \hookrightarrow X$. In fact, any computable homeomorphism is a computable embedding.

Example 2.2.20. Let $h\colon B \to \mathbb{R}^n$ ($B = B(0,1)$) be the computable homeomorphism of example 2.2.9 and $p_0\colon \mathbb{R}^n \to \mathbb{R}^{n+1}$ be such that $p_0(x_1, \ldots, x_n) = (0, x_1, \ldots, x_n)$. Clearly p_0 is a computable embedding of $\mathbf{R}^n$ into $\mathbf{R}^{n+1}$. We can compose h with p_0 to obtain a computable embedding $B(0,1) \hookrightarrow \mathbb{R}^{n+1}$ of $B_{\mathbf{R}^n}$ into $\mathbf{R}^{n+1}$.

2.3 Computably Hausdorff spaces

In the theory of computable topology, computable versions of the standard separation axioms T_i ($i = 0, 1, 2$) has been proposed [GWX08, Wei10]. Some of the relationships between these computable axioms and many examples are studied in [Wei10]. One of this axioms will be used in this thesis, which is one of the computable variants of the T_2 (Hausdorff) separation axiom. A space (X, τ) is T_2 or *Hausdorff* if and only if the following condition holds:

$$(\forall x, y \in X)(x \neq y \Rightarrow (\exists U, V \in \tau) U \cap V = \varnothing \wedge x \in U \wedge y \in V).$$

Many examples of Hausdorff spaces exist in the literature [Arm83, Eng89, Mun00]. The next definition is a constructive version of the Hausdorff property.

Definition 2.3.1. A computable topological space $\mathbf{X} = (X, \tau, \beta, \nu)$ is called *computably Hausdorff* if there exists a c.e. set $H \subseteq \operatorname{dom} \nu \times \operatorname{dom} \nu$ such that

$$(\forall (u, v) \in H)(\nu(u) \cap \nu(v) = \varnothing), \tag{2.17}$$

$$(\forall x, y \in X)(x \neq y \Rightarrow (\exists (u, v) \in H)(x \in \nu(u) \wedge y \in \nu(v))). \tag{2.18}$$

Example 2.3.2. The computable euclidean space $\mathbf{R}^n$ is a computably Hausdorff space. We claim that the set H defined as

$$H \overset{\text{def}}{=} \{(u, v) \in \Sigma^* \times \Sigma^* \mid \mu^n(u) \cap \mu^n(v) = \varnothing\}$$

is c.e. and satisfies Equations (2.17) and (2.18). By definition, H fulfills (2.17), to prove that H is c.e., we can use Theorem 1.2.6. To see that H satisfies (2.18), we proceed as follows. As $\mathbb{R}^n$ is Hausdorff, then given any two points $x, y \in \mathbb{R}^n$ with $x \neq y$, there exist open sets $U, V \in \tau^n$ such that $U \cap V = \varnothing$ and $x \in U, y \in V$. Thus we can find two base elements $B_1, B_2 \in \beta^n$ such that $x \in B_1 \subseteq U$ and $y \in B_2 \subseteq V$. As μ^n is a notation, it is surjective, so that there exist $u_1, u_2 \in \operatorname{dom} \mu^n$ with $\mu^n(u_i) = B_i$, $i = 1, 2$. Because $U \cap V = \varnothing$, $\mu^n(u_1) \cap \mu^n(u_2) = \varnothing$, so that $u_1, u_2 \in H$. Therefore H satisfies Definition 2.3.1 and $\mathbf{R}^n$ is a computably Hausdorff space.

The computable Hausdorff axiom of Definition 2.3.1 is the strongest of all the computable separation axioms of [Wei10]. Of course, A computably Hausdorff space is a Hausdorff space in the usual sense. We list some properties of computably Hausdorff topological spaces [GWX08, Wei10].

Theorem 2.3.3. *Let $\mathbf{X} = (X, \tau, \beta, \nu)$ be a computable topological space.*

1. *If $\mathbf{X}$ is Hausdorff and the set $\{(u, v) \in \operatorname{dom} \nu \times \operatorname{dom} \nu \mid \nu(u) \cap \nu(v) = \varnothing\}$ is c.e., then $\mathbf{X}$ is computably Hausdorff.*
2. *If $\mathbf{X}$ is computably Hausdorff and $A \subseteq X$, then the computable subspace $A_{\mathbf{X}}$ is computably Hausdorff.*
3. *The following are equivalent*
   a) *$\mathbf{X}$ is computably Hausdorff.*
   b) *The set $\{(x, y) \in X \times X \mid x \neq y\}$ is (δ, δ)-c.e.*
   c) *$\delta \leq \delta^-$.*
4. *If $\mathbf{X}$ is computably Hausdorff then $\kappa \leq \psi^-$.*
5. *If $\mathbf{Y} = (Y, \tau', \beta', \nu')$ is another computable topological space and $\mathbf{X}, \mathbf{Y}$ are computably Hausdorff, then the computable product space $\overline{\mathbf{X}} = (X \times Y, \overline{\tau}, \overline{\beta}, \overline{\nu})$ is computably Hausdorff.*

More information about Computably Hausdorff spaces and many other computable separation axioms, results, examples and counterexamples can be found in [Wei10]. The computable Hausdorff property will be very important for our work on Computable manifolds, as it is important the standard Hausdorff property for topological manifolds.

3 Computable manifolds

In this chapter, we present the concept of *computable manifold*. The first thing to do is to give the definition of what we call *computable structure*. This is very similar to the way smooth and PL manifolds are defined, by constructing smooth and PL structures for topological manifolds. Once we have defined computable manifolds, we will talk about computable functions between two computable manifolds and prove some properties about them.

In Section 3.1, we begin introducing one of the key components of the definition of computable manifolds: *Computable atlases*. This important definition is given in terms of *predicate spaces* (see Section 2.1.2) and give many examples of topological spaces equipped with computable atlases and after that, we use computable atlases to define *computable structures*. We show how a computable structure is characterized by a computable topology and then we define computable manifolds. Examples of computable manifolds are given at the end of the section. In Section 3.2, we show that computable functions are well suited to be used as morphisms between computable manifolds and also we study computable homeomorphisms between computable manifolds. Finally, in Section 3.3, we prove basic properties of computable manifolds. While some of these properties are only valid for computable manifolds (e.g., Lemmas 3.3.2 and 3.3.3), other properties are computable versions of known basic properties of topological manifolds, like Theorem 3.3.5.

3.1 Computable structures on topological manifolds

We first introduce the concept of *computable atlas* and prove some facts about it. We show that, as an ordinary topological atlas on a set X induces a topology on X, a computable atlas on X induces a computable topology on X and we prove that this topology characterizes the computable structure determined by the computable atlas. Throughout this section, we workout many examples of computable atlases and computable manifolds. We also show that the product of two computable manifold is a computable manifold, and the computable topology of the product manifold is the product topology of the two factor manifolds.

3.1.1 The computable predicate space induced by a topological atlas

Assume that X is a non-empty set and that $\Phi = \{(\varphi_i, U_i)\}_{i\in I}$ is an n-dimensional atlas on X with index set $I \subseteq \Sigma^*$ (so that Φ is countable). If $\varphi_i \colon U_i \to \varphi_i[U_i]$ is a chart, then by Definition 1.1.16, $\varphi_i[U_i] \subseteq \mathbb{R}^n_+$ is an open set,thus

$$\varphi_i[U_i] = \bigcup_{w\in S_i} \mu^n_+(w),$$

where $S_i = \{w \in \operatorname{dom}(\mu_+^n) \mid \mu_+^n(w) \subset \varphi_i[U_i]\}$. Applying φ_i^{-1} we get that $U_i = \bigcup_{w \in S_i} \varphi_i^{-1}[\mu_+^n(w)] \subset X$. Let A_Φ be the set $A_\Phi = \{(i,w) \mid i \in \Sigma^* \text{ and } w \in \operatorname{dom}(\mu_+^n)\} \subseteq \Sigma^* \times \Sigma^*$ and define the set of strings $\Lambda_\Phi \subseteq \Sigma^*$ as

$$\Lambda_\Phi = \{\langle i,w\rangle \mid (i,w) \in A_\Phi\},$$

that is, Λ_Φ is just the image of the set A_Φ under the function $\langle\cdot,\cdot\rangle$ (see Definition 1.2.3 and Equation (1.1)). For each $\langle i,w\rangle \in \Lambda_\Phi$, define the set $B_{\langle i,w\rangle} \subseteq X$ by

$$B_{\langle i,w\rangle} = \begin{cases} \varphi_i^{-1}[\mu_+^n(w)] & \text{if } i \in I \text{ and } \mu_+^n(w) \subseteq \varphi_i[U_i], \\ \varnothing & \text{otherwise.} \end{cases} \tag{3.1}$$

Each set $B_{\langle i,w\rangle}$ with $\langle i,w\rangle \in \Lambda_\Phi$ is called a *computable ball (with boundary, if $B_{\langle i,w\rangle} \neq \varnothing$ and $\mu_+^n(w) \cap \partial\mathbb{R}_+^n \neq \varnothing$)*. Notice that unlike an ordinary ball, a computable ball can be empty. Because the set $\{U_i\}_{i\in I}$ covers X, we can see that the set $\mathcal{B}_\Phi = \{B_j \mid j \in \Lambda_\Phi\}$ also covers X. Let the notation $\lambda_\Phi \colon \subseteq \Sigma^* \to \mathcal{B}_\Phi$ be given as follows: $\operatorname{dom}\lambda_\Phi = \{z \in \Sigma^* \mid z = \langle i,w\rangle \text{ and } \langle i,w\rangle \in \Lambda_\Phi\}$ and

$$\lambda_\Phi(\langle i,w\rangle) = B_{\langle i,w\rangle}.$$

Proposition 3.1.1. *The triple $\mathbf{Z}_\Phi = (X, \mathcal{B}_\Phi, \lambda_\Phi)$ is a computable predicate space.*

Proof. Clearly, the set $\operatorname{dom}\lambda_\Phi = \Lambda_\Phi$ is a computable set, because $\Sigma^* \times \operatorname{dom}(\mu_+^n)$ is computable, so we only need to prove that for $x, y \in X$

$$\mathcal{P}_x(X) = \mathcal{P}_y(X) \Rightarrow x = y.$$

Assume that $\mathcal{P}_x(X) = \mathcal{P}_y(X)$, so that there is a string $\langle j,u\rangle \in \Sigma^*$ such that $x, y \in B_{\langle j,u\rangle}$. For $z \in \{x,y\}$, define the set

$$\mathcal{P}_z^j = \{B_{\langle j,w\rangle} \mid B_{\langle j,w\rangle} \in \mathcal{P}_z(X)\}$$

and using φ_j we can define the set

$$\varphi_j(\mathcal{P}_z^j) = \{\mu_+^n(w) \mid \mu_+^n(w) \subset \varphi_j[U_j] \wedge \varphi_j(z) \in \mu_+^n(w)\} \subset \beta^n,$$

then using our hypothesis, we can prove that $\mathcal{P}_x^j = \mathcal{P}_y^j$ and clearly this implies that $\varphi_j(\mathcal{P}_x^j) = \varphi_j(\mathcal{P}_y^j)$.

Let $\mathbf{Z}_+^n = (\mathbb{R}_+^n, \beta_+^n, \mu_+^n)$ be the standard computable predicate space[1] associated with $\mathbb{R}_+^n$ (see Example 2.1.17) and $V \in \mathcal{P}_{\varphi_j(x)}(\mathbb{R}_+^n)$. V is an open neighborhood of $\varphi_j(x)$ and for a suitable $u \in \operatorname{dom}(\mu_+^n)$, we can choose $\mu_+^n(u) \in \varphi_j(\mathcal{P}_x^j)$ such that $\varphi_j(x), \varphi_j(y) \in \mu_+^n(u) \subseteq V \cap \varphi_j[U_j]$, thus $V \in \mathcal{P}_{\varphi_j(y)}(\mathbb{R}_+^n)$. This shows that $\mathcal{P}_{\varphi_j(x)}(\mathbb{R}_+^n) \subset \mathcal{P}_{\varphi_j(y)}(\mathbb{R}_+^n)$ and a similar argument can be used to check that the other inclusion holds. Therefore in the predicate space $\mathbf{Z}_+^n$,

$$\mathcal{P}_{\varphi_j(x)}(\mathbb{R}_+^n) = \mathcal{P}_{\varphi_j(y)}(\mathbb{R}_+^n),$$

implying that $\varphi_j(x) = \varphi_j(y)$ and since φ_j is injective, $x = y$. We have proven that the triple $\mathbf{Z}_\Phi = (X, \mathcal{B}_\Phi, \lambda_\Phi)$ is a computable predicate space. $\square$

[1]This predicate space is defined for $\mathbb{R}_+^n$ in a similar way as $\mathbf{Z}^n$ is contructed for $\mathbb{R}^n$ in Example 2.1.17.

Remark. Since an atlas Φ on a set X induces a computable predicate space $\mathbf{Z}_\Phi$ on X, Φ induces a structure of computable topological space on X. By Lemma 2.1.16, the effective space

$$\mathbf{T}_\Phi(X) = T(\mathbf{Z}_\Phi)$$

is a computable topological space, such that the topology induced by $\mathbf{Z}_\Phi$ is precisely the topology τ_Φ induced by Φ. But the set $\mathcal{B}_\Phi$ of computable balls is not only a subbase, but a base of τ_Φ, so that we have another effective topological space $\mathbf{X}_\Phi = (X, \tau_\Phi, \mathcal{B}_\Phi, \lambda_\Phi)$ associated with X. We define the *computable topological space associated to* Φ (and induced on X) as the space $\mathbf{T}_\Phi(X)$. In general, we cannot use the effective space $\mathbf{X}_\Phi$, because it could happen that it is not computable. But if it is the case that $\mathbf{X}_\Phi$ is computable, then by Lemma 2.2.17, $\mathbf{X}_\Phi$ is equivalent to $\mathbf{T}_\Phi(X)$. If $\delta\colon \subseteq \Sigma^\omega \to X$ is the inner representation of X induced by $\mathbf{T}_\Phi(X)$, then by part 2. of Lemma 2.1.16, $\delta \sim \delta_{\mathbf{Z}_\Phi}$, hence for computability matters, we can interchange δ with $\delta_{\mathbf{Z}_\Phi}$. Moreover, we will use the symbol δ_Φ to denote any of the representations δ and $\delta_{\mathbf{Z}_\Phi}$. In fact, because the computable space $\mathbf{T}_\Phi(X)$ depends on the atlas Φ, all the elements of $\mathbf{T}_\Phi(X)$ and all notations and representations induced by this space will be denoted with a "Φ" subindex, so that $\mathbf{T}_\Phi(X) = (X, \tau_\Phi, \beta_\Phi, \nu_\Phi)$, where τ_Φ is the topology induced by Φ on X; β_Φ is the set of all finite intersections of elements of $\mathcal{B}_\Phi$; ν_Φ is the notation of β_Φ induced by λ_Φ and $\delta_\Phi, \theta_\Phi, \psi_\Phi, \kappa_\Phi$, etc., are the representations given in Definitions 2.1.6 and 2.1.7 respectively, for the computable space $\mathbf{T}_\Phi(X)$.

3.1.2 Computable atlases

We are now ready to formulate our definition of *computable manifold*. We start by defining what we call *computable atlas*. Remember that the computable topological spaces $\mathbf{R}^n = (\mathbb{R}^n, \tau^n, \beta^n, \mu^n)$ and $\mathbf{R}^n_+ = (\mathbb{R}^n_+, \tau^n_+, \beta^n_+, \mu^n_+)$ from Examples 2.1.2 and 2.1.19 are called computable euclidean space and computable half space respectively, of dimension n, and that each standard representation induced by $\mathbf{R}^n$ ($\mathbf{R}^n_+$) is denoted by γ^n (γ^n_+).

Definition 3.1.2. An n-dimensional *computable atlas* on a set X is a topological atlas[2] $\Phi = \{(\varphi_i, U_i)\}_{i\in I}$ ($I \subseteq \Sigma^*$) such that the following properties are satisfied:

(a) For $i \in I$, the map $\varphi_i\colon U_i \subseteq X \to \mathbb{R}^n_+$ is a $(\delta_{\mathbf{Z}_\Phi}, \delta^n_+)$-computable function and the inverse $\varphi_i^{-1}\colon \varphi_i[U_i] \to U_i$ is $(\delta^n_+, \delta_{\mathbf{Z}_\Phi})$-computable.

(b) Each set $\varphi_i[U_i] \subseteq \mathbb{R}^n_+$ is a θ^n_+-computable subset of $\mathbb{R}^n_+$.

Notice that we have formulated our formal definition of computable atlases, using as a basis the concept of atlases that are specified in Definition 1.1.16, which use open subsets of the half space $\mathbb{R}^n_+$. We need to do this in order to be able to define computable atlases and computable structures for general topological manifolds with boundary. However, when we deal with manifolds with empty boundary, any chart $\varphi\colon U \to \mathbb{R}^n_+$ of a manifold M has its image $\varphi[U]$ entirely contained in the interior of $\mathbb{R}^n_+$ and can be thought as an open set of euclidean space $\mathbb{R}^n$. Thus, if we are using manifolds without boundary, we can replace the computable half space $\mathbf{R}^n_+$ with computable euclidean space $\mathbf{R}^n$ in all the definitions and results that we will obtain for computable manifolds. Therefore, from now on, we adopt the following convention: If we work with topological manifolds without boundary, we will use the computable space $\mathbf{R}^n$ instead of $\mathbf{R}^n_+$ to apply definitions and results of computable manifolds. We now present some examples of computable atlases.

[2]See Subsection 1.1 and Definition 1.1.16.

Example 3.1.3. Let $n \geqslant 0$. The identity $1_{\mathbb{R}^n} \colon \mathbb{R}^n \to \mathbb{R}^n$ is a (δ^n, δ^n)-computable function which covers $\mathbb{R}^n$ and this is clearly a θ^n-computable open set, thus it determines an n-dimensional computable atlas $\Phi = \{(1_{\mathbb{R}^n}, \mathbb{R}^n)\}$ on $\mathbb{R}^n$.

Example 3.1.4. The map $\varphi_1 \colon \mathbb{R} \to \mathbb{R}$ defined by $\varphi_1(x) = x + a$ $(a \in \mathbb{R})$ is a homeomorphism of the line onto itself. We now prove that $\Psi = \{(\varphi_1, \mathbb{R})\}$ is a computable atlas on $\mathbb{R}$. Clearly, $\mathbb{R}$ is a θ^1-computable open set in $\mathbb{R}$, now we need to show that φ_1 and its inverse are computable with respect to δ_Ψ and δ^1. φ_1 is (δ_Ψ, δ^1)-computable, because given $x \in \mathbb{R}$ and $p \in \operatorname{dom} \delta_\Psi$ such that $\delta_\Psi(p) = x$, we have that

$$\delta_\Psi(p) = x \iff (\forall w \in \Sigma^*)(w \ll p \Leftrightarrow w \in \operatorname{dom} \lambda_\Psi \text{ and } x \in \lambda_\Psi(w)),$$

So, for each $w \ll p, x \in \lambda_\Psi(w) = \varphi_1^{-1}\big[\mu^1(z)\big]$ with $w = \langle 1, z \rangle$. Therefore, $\varphi_1(x) \in \mu^1(z)$ for all $w \ll p$. Let $\mathfrak{M}$ be a TTE machine with the following program. On the input $p \in \Sigma^\omega$

```
1. For each $w = \langle 1, z \rangle \ll p$
    1.1 output $\iota(z)$
```

By the previous argument, the machine $\mathfrak{M}$ outputs a string $q \in \Sigma^\omega$ such that $(\forall z \in \Sigma^*)(z \ll q \Leftrightarrow z \in \operatorname{dom}(\mu^1)$ and $\varphi_1(x) \in \mu^1(z))$. By definition 2.1.6, $\delta^1(q) = \varphi_1(x)$, thus $\mathfrak{M}$ computes a function which realizes φ_1 with respect to δ_Ψ and δ^1. To show that φ_1^{-1} is (δ^1, δ_Ψ)-computable, the argument is very similar, we omit it. We have shown that Ψ is a 1-dimensional computable atlas on $\mathbb{R}$.

Notice that the computability of the atlas Ψ is independent of the computability of the real number a (with respect to δ^1), given in the definition of φ_1. We will come back to this example later.

Example 3.1.5. We prove that the atlas Φ for $\mathbb{S}^1 \subset \mathbb{R}^2$ given in Example 1.1.15 is computable. Φ is defined by the pairs

$$\Phi = \{(f_+, U_+), (f_-, U_-), (g_+, V_+), (g_-, V_-)\}.$$

To check that Φ is computable, we need to show that each chart $\varphi \colon U \to \mathbb{R}$ in Φ is a $(\delta_{\mathbf{Z}_\Phi}, \delta^1)$-computable function with $(\delta^1, \delta_{\mathbf{Z}_\Phi})$-computable inverse and the sets $\varphi\big[U\big]$ are θ^1-computable open subsets of $\mathbb{R}$. Each set $\varphi\big[U\big]$ is clearly a θ^1-computable open subset of $\mathbb{R}$, because we have that $\varphi\big[U\big] = (-1, 1)$ and the latter set is θ^1-computable. Now we give the full proof of the computability of f_+ and its inverse, the other cases f_-, g_+ and g_- are very similar.

Let $f_1 = f_+, f_2 = f_-, f_3 = g_+$ and $f_4 = g_-$ To see that f_1 is $(\delta_{\mathbf{Z}_\Phi}, \delta^1)$-computable, consider the following Type-2 machine $\mathfrak{F}_1$. On input $p \in \operatorname{dom} \delta_{\mathbf{Z}_\Phi}$ $(\delta_{\mathbf{Z}_\Phi}(p) = (x, y) \in \mathbb{S}^1)$:

```
1. For each $z = \langle i, w \rangle \ll p$,
    1.1 if $i = 1$, then output $\iota(w)$;
    1.2 Otherwise, execute the following:
      1.2.1 Compute $a_y, b_y$ such that $y \in (a_y, b_y) \subset (-1, 1)$;
      1.2.2 compute $a_x = \sqrt{1 - a_y^2}$;
      1.2.3 compute $b_x = \sqrt{1 - b_y^2}$;
```

1.2.4 compute w' such that $(b_x, a_x) \subseteq \mu^1(w') \subset (-1,1)$;
1.2.5 output $\iota(w')$.

We claim that $\mathfrak{F}_1$ computes a function $F_1\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ which realizes f_1. We first check that each step of $\mathfrak{F}_1$ can be done in finite time. Step 1.1 is clearly computable, and for step 1.2, we only need to check that steps 1.2.1-1.2.5 can be calculated in finite time by $\mathfrak{F}_1$. First of all, if $\mathfrak{F}_1$ is executing step 1.2, then we have that $i \neq 1$, so that the string $z = \langle i, w\rangle$ represents a computable ball $f_i^{-1}\left[\mu^1(w)\right]$ such that $f_i \neq f_1$. Remember that f_1 is defined on the set $U_+ = \{(x,y) \in \mathbb{S}^1 \mid y > 0\}$, so that if $f_i \neq f_1 = f_+$, then $f_i = f_3$ or $f_i = f_4$. Without loss of generality, assume that $f_i = f_3$.

Step 1.2.1 can be computed in finite time, because since $(x,y) = \delta_{\mathbf{Z}_\Phi}(p) \in f_3^{-1}\left[\mu^1(w)\right]$, then $f_3(x,y) = g_+(x,y) = y \in \mu^1(w) \subset (-1,1)$. Using the string $w \in \operatorname{dom}(\mu^1)$, $\mathfrak{F}_1$ can compute $a_y, b_y \in \mathbb{Q}$ with $y \in (a_y, b_y) \subset \mu^1(w)$, thus step 1.2.1, can be done in finite time by $\mathfrak{F}_1$. Steps 1.2.2 and 1.2.3 are computable, because a_y, b_y are rationals and by part (b) of Lemma 2.2.5, the square root is a computable function. Step 1.2.4 can be calculated by $\mathfrak{F}_1$, because by the previous steps, a_x and b_x are computable points in $\mathbb{R}$ and the given set inclusions can be tested by using Theorem 1.2.6. Step 1.2.5 is clearly computable.

Now we prove the correctness of $\mathfrak{F}_1$. Let $(x,y) = \delta_{\mathbf{Z}_\Phi}(p) \in \operatorname{dom} f_1$. and take $\langle i, w\rangle \ll p$, so that $(x,y) \in f_i^{-1}\left[\mu^1(w)\right]$. If $i = 1$ then $f_1(x,y) \in \mu^1(w)$ and in this case, $\mathfrak{F}_1$ outputs $\iota(w)$ in the output tape. When $i \neq 1$, $i = 3$ or $i = 4$, thus $f_i(x,y) = y \in \mu^1(w)$. There exist $a_y, b_y \in \mathbb{Q}$ such that $y \in \mu^1(w) = (a_y, b_y)$. Since f_i is bijective, $a_x = f_i^{-1}(a_y)$ and $b_x = f_i^{-1}(b_y)$ are defined and they satisfy the equations

$$a_x = \sqrt{1 - a_y^2};$$

$$b_x = \sqrt{1 - b_y^2};$$

(notice that $b_x < a_x$, because $a_y < b_y$) and it is immediate to show that $b_x < f_1(x,y) < a_x$. There exist rational numbers a, b such that $f_1(x,y) \in (b_x, a_x) \subseteq (a,b) \subset (-1,1)$ and the set (a,b) is represented by a string $w' \in \operatorname{dom}(\mu^1)$, this string is computed by $\mathfrak{F}_1$ in step 1.2.5, thus the output $\iota(w')$ is correct. Therefore the machine $\mathfrak{F}_1$ computes a function F such that on the input $p \in \operatorname{dom} \delta_{\mathbf{Z}_\Phi}$, $F(p)$ satisfies

$$(\forall w' \in \Sigma^*)(w' \ll F(p) \Leftrightarrow w' \in \operatorname{dom}(\mu^1) \text{ and } f_1(x,y) \in \mu^1(w')).$$

Therefore $\delta^1(F(p)) = f_1(x,y)$ and F realizes f_1 with respect to $\delta_{\mathbf{Z}_\Phi}$ and δ^1, so that f_1 is $(\delta_{\mathbf{Z}_\Phi}, \delta^1)$-computable.

It only remains to prove that f_1^{-1} is $(\delta^1, \delta_{\mathbf{Z}_\Phi})$-computable. There exists a Type-2 machine $\mathfrak{G}_1$ that, on input $q \in \operatorname{dom} \delta^1$, does the following: For each $z \ll q$, $\mathfrak{G}_1$ checks if $\mu^1(z) \subseteq (-1,1)$, if so, then it writes $\iota(\langle 1, z\rangle)$ on the output tape; otherwise $\mathfrak{G}_1$ ignores the string z. It is easy to see that $\mathfrak{G}_1$ computes a function $G\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ which realizes f_1^{-1} with respect to δ^1 and $\delta_{\mathbf{Z}_\Phi}$.

Therefore, the atlas Φ fulfills Definition 3.1.2 so that it is a 1-dimensional computable atlas for the circle.

We now show that with a computable atlas, the transition functions satisfy the expected computability properties inside the induced computable topological space $\mathbf{T}_\Phi(X)$.

Lemma 3.1.6. *Let $\Phi = \{(\varphi_i, U_i)\}_{i\in I}$ be an n-dimensional computable atlas on X. Then for the computable spaces $\mathbf{T}_\Phi(X)$ and $\mathbf{R}^n_+$:*

(i) For all $i \in I$, the chart $\varphi_i \colon U_i \to \varphi_i\big[U_i\big] \subseteq \mathbb{R}^n_+$ is a computable homeomorphism (with respect to δ_Φ and δ^n_+) and U_i is a θ_Φ-computable open set in X.

(ii) For each $i, j \in I$, $U_i \cap U_j$ is θ_Φ-computable open in X and $\varphi_i\big[U_i \cap U_j\big]$ is θ^n_+-computable open in $\mathbb{R}^n_+$.

(iii) Each transition function $\varphi_i\varphi_j^{-1}$ is a computable homeomorphism between θ^n_+-computable subsets of $\mathbb{R}^n_+$.

Proof. (i) By Proposition 1.1.10, each φ_i is an homeomorphism onto its image $\varphi_i\big[U_i\big]$ and that φ_i (φ_i^{-1}) is computable is true because φ_i (φ_i^{-1}) is $(\delta_{\mathbf{Z}_\Phi}, \delta^n_+)$-computable ($(\delta^n_+, \delta_{\mathbf{Z}_\Phi})$-computable) and by Lemma 2.1.16, $\delta_{\mathbf{Z}_\Phi} \sim \delta_\Phi$. U_i is θ_Φ-computable because since φ_i is computable, the map $W \mapsto \varphi_i^{-1}\big[W\big]$ is $(\theta^n_+, \theta_\Phi)$-computable (Definition 2.2.4); (ii) Follows by combining (i) and part 1. of Theorem 2.1.13 with Definition 2.2.4; (iii) is immediate because composition of computable functions between computable topological spaces is again, computable. $\square$

We continue with more examples of computable atlases.

Example 3.1.7. It is easy to see that the topology induced by the computable atlas $\{(1_{\mathbb{R}^n}, \mathbb{R}^n)\}$ on $\mathbb{R}^n$ of Example 3.1.3 is of course, the usual euclidean topology and clearly $\mathbf{T}_\Phi(\mathbb{R}^n)$ is equivalent to $\mathbf{R}^n$. Consider now the atlas $\Phi = \{(f_+, U_+), (f_-, U_-), (g_+, V_+), (g_-, V_-)\}$ on $\mathbb{S}^1$ of Example 3.1.5. We have proven that it is computable. But which is the topology that Φ induces on $\mathbb{S}^1$? Let $\mathbf{S}^1$ be the computable 1-sphere of $\mathbf{R}^2$ introduced in Example 2.2.10.

Lemma 3.1.8. *The computable topological spaces $\mathbf{T}_\Phi(\mathbb{S}^1)$ and $\mathbf{S}^1$ are equivalent.*

Proof. Notice that for suitable $a, b, c, d, e, f \in \mathbb{Q}$ and $\varphi, \psi, \alpha \in \{f_+, f_-, g_+, g_-\}$ the condition

$$\varphi^{-1}\big[(a,b)\big] \subseteq \psi^{-1}\big[(c,d)\big] \cap \alpha^{-1}\big[(e,f)\big],$$

can be verified algorithmically. For example, $g_-^{-1}\big[(a,b)\big] \subseteq f_+^{-1}\big[(c,d)\big] \cap g_+^{-1}\big[(e,f)\big]$ is equivalent to the condition

$$(\forall (x,y) \in \mathbb{R}^2)(x^2 + y^2 = 1 \wedge x < 0 \wedge y \in (a,b) \wedge (a,b) \subset (-1,1) \Rightarrow y > 0 \wedge x \in (c,d) \\ \wedge (c,d) \subset (-1,1) \wedge x > 0 \wedge y \in (e,f) \wedge (e,f) \subset (-1,1)).$$

By Theorem 1.2.6, all expressions of this kind can always be checked in finite time by a Turing machine. Hence, the effective space[3] $\mathbf{S}^1_\Phi = (\mathbb{S}^1, \tau_\Phi, \mathcal{B}_\Phi, \lambda_\Phi)$ becomes a computable topological space and by Lemma 2.2.17, $\mathbf{S}^1_\Phi$ is equivalent to $\mathbf{T}_\Phi(\mathbb{S}^1)$. We now show that $\mathbf{S}^1_\Phi$ is equivalent to the computable subspace $\mathbf{S}^1$. By Definition 2.2.12, we need to show that $\lambda_\Phi \leq \theta^2_{\mathbb{S}^1}$ and $\mu^2_{\mathbb{S}^1} \leq \theta_\Phi$.

To see that $\lambda_\Phi \leq \theta^2_{\mathbb{S}^1}$, it is enough to build a Type-2 machine $\mathfrak{M}$ which, on input $z \in \operatorname{dom} \lambda_\Phi$, enumerates all string $w \in \operatorname{dom}(\mu^2)$ and for each w, $\mathfrak{M}$ checks if

$$\mu^2_{\mathbb{S}^1}(w) = \mu^2(w) \cap \mathbb{S}^1 \subseteq \lambda_\Phi(z), \tag{3.2}$$

if so, then $\mathfrak{M}$ outputs $\iota(w)$ on the output tape; otherwise it continues with the next enumerated string of $\operatorname{dom}(\mu^2)$. For each w, $\mathfrak{M}$ can perform the test given in Equation (3.2) by generating an elementary expression equivalent to (3.2). For example, if $\mu^2(w) = B(q,r)$ $(q = (q_1, q_2))$ and

[3]See the remark after the proof of Proposition 3.1.1.

$\lambda_\Phi(z) = g_+^{-1}[\mu^1(z_1)]$, with $\mu^1(z_1) = (a, b) \subseteq (-1, 1)$, then an elementary expression equivalent to (3.2) is the following

$$(\forall x, y)(x^2 + y^2 = 1 \text{ and } (x - q_1)^2 + (y - q_2)^2 < r^2 \Rightarrow a < y \text{ and } y < b).$$

By Theorem 1.2.6, $\mathfrak{M}$ can check this expression in finite time. All other cases are very similar. Thus the function computed by $\mathfrak{M}$ outputs an infinite string $q \in \Sigma^\omega$ such that

$$\theta^2_{\mathbb{S}^1}(q) = \bigcup_{w \ll q} \mu^2_{\mathbb{S}^1}(w) = \lambda_\Phi(z),$$

and the output of the machine is correct, thus $\lambda_\Phi \leq \theta^2_{\mathbb{S}^1}$. The argument for the case $\mu^2_{\mathbb{S}^1} \leq \theta_\Phi$ is almost identical, so we omit it. Therefore $\mathbf{S}^1_\Phi$ is equivalent to the computable subspace $\mathbf{S}^1$ and since $\mathbf{S}^1_\Phi$ is equivalent to $\mathbf{T}_\Phi(\mathbb{S}^1)$, we have that $\mathbf{S}^1$ is equivalent to $\mathbf{T}_\Phi(\mathbb{S}^1)$, so that the computable topology on $\mathbb{S}^1$ induced by Φ is the computable subspace topology of $\mathbb{S}^1$. □

Example 3.1.9. An atlas can be defined for the sphere $\mathbb{S}^n \subset \mathbb{R}^{n+1}$ with the stereographic projection. Let $P_1 = (0, \ldots, 0, 1), P_{-1} = (0, \ldots, 0, -1)$ and define $U_r = \mathbb{S}^n - \{P_r\}$ with $r = 1, -1$. $U_1 \cup U_{-1} = \mathbb{S}^n$ and if we define $s_1 \colon U_1 \to \mathbb{R}^n$, $s_1(x, t) = \frac{x}{1-t}$ and $s_{-1} \colon U_{-1} \to \mathbb{R}^n$, $s_{-1}(x, t) = \frac{x}{1+t}$ with $x = (x_1, \ldots, x_n)$. Then $\Psi = \{(s_1, U_1), (s_{-1}, U_{-1})\}$ is a topological atlas for $\mathbb{S}^n$. We prove that Ψ is a computable atlas for $\mathbb{S}^n$ as follows: The atlas Ψ induces the effective spaces[3] $\mathbf{S}^n_\Psi = (\mathbb{S}^n, \tau_\Psi, \mathcal{B}_\Psi, \lambda_\Psi)$ and $\mathbf{T}_\Psi(\mathbb{S}^n)$, being the latter computable. Now, since the maps s_1, s_{-1} are rational functions with coefficients in $\mathbb{Q}$, we can apply Theorem 1.2.6 to deduce that the decision problem

$$s_r^{-1}[B_1] \subseteq s_t^{-1}[B_2] \cap s_u^{-1}[B_3], \quad B_i \in \beta^n, r, t, u \in \{1, -1\},$$

is computable, thus $\mathbf{S}^n_\Psi = (\mathbb{S}^n, \tau_\Psi, \mathcal{B}_\Psi, \lambda_\Psi)$ is a computable topological space and by Lemma 2.2.17, it is equivalent to $\mathbf{T}_\Psi(\mathbb{S}^n)$. Our next step is to show that $\mathbf{S}^n_\Psi$ and $\mathbf{S}^n = \mathbb{S}^n_{\mathbf{R}^{n+1}}$ are equivalent computable spaces. This can be done easily using Definition 2.2.12 and Theorem 1.2.6, just as we did in Lemma 3.1.8. If $\mu^{n+1}_{\mathbb{S}^n}$ is the notation for base elements of $\mathbf{S}^n$ and θ_Ψ is the representation for open sets of $\mathbf{S}^n_\Psi$, then to prove that $\mu^{n+1}_{\mathbb{S}^n} \leq \theta_\Psi$, we have that $s_r^{-1}[\mu^n(z)] \subseteq \mu^{n+1}_{\mathbb{S}^n}(w)$ ($z \in \mathrm{dom}(\mu^n)$ and $w \in \mathrm{dom}(\mu^{n+1})$) is equivalent to the expression

$$(\forall y \in \mathbb{R}^{n+1})(y \in \mathbb{S}^n \text{ and } s_r(y) \in \mu^n(z) \Rightarrow y \in \mathbb{S}^n \cap \mu^{n+1}(w)) \tag{3.3}$$

and this expression can be easily translated into an elementary expression. With all this data, a Type-2 machine can be constructed such that, on input $w \in \mathrm{dom}(\mu^{n+1}_{\mathbb{S}^n})$, enumerates all pairs (r, z) ($r \in \{-1, 1\}, z \in \mathrm{dom}(\mu^n)$) and test if w, r and z satisfies (3.3), if this is the case, then the machine outputs $\iota(z)$ on the output tape. This machine computes a function which translates $\mu^{n+1}_{\mathbb{S}^n}$-names into θ_Ψ-names, that is, $\mu^{n+1}_{\mathbb{S}^n} \leq \theta_\Psi$. To prove that $\lambda_\Psi \leq \theta^{n+1}_{\mathbb{S}^n}$, the argument is almost the same. Thus $\mathbf{S}^n_\Psi$ and $\mathbf{S}^n$ are equivalent, therefore $\mathbf{T}_\Psi(\mathbb{S}^n)$ and $\mathbf{S}^n$ are equivalent computable spaces.

Now to show that the atlas Ψ is computable, the argument is the following: Since $\mathbf{T}_\Psi(\mathbb{S}^n)$ and $\mathbf{S}^n$ are equivalent, $\delta_\Psi \sim \delta^{n+1}_{\mathbb{S}^n}$, so that to check that s_1, s_{-1} and their inverses are computable with respect to δ_Ψ and δ^n, it is enough to show that they are computable with respect to $\delta^{n+1}_{\mathbb{S}^n}$ and δ^n. But $\delta^{n+1}_{\mathbb{S}^n}$ is simply the representation δ^{n+1} of $\mathbb{R}^{n+1}$, restricted to $\mathbb{S}^n$ (see Definition 1.2.2). In summary, we only need to show that the charts are computable with respect to δ^{n+1} and δ^n. But the maps

s_1, s_{-1} and their inverses are defined in terms of sums, multiplications and square roots, thus we only need to apply Lemma 2.2.5 and we are done.

To finish the proof, we need to show that the sets $s_i[U_i]$ are θ^n-computable in $\mathbb{R}^n$. But this is immediate, because $s_i[U_i] = \mathbb{R}^n$. Therefore Ψ is a computable atlas for $\mathbb{S}^n$.

Example 3.1.10. Let $\mathbb{R}\mathrm{P}^n$ be n-dimensional real projective space, the set of all 1-dimensional vector subspaces of $\mathbb{R}^{n+1}$. Each subspace is spanned by a non-zero vector $v = (x_1, \dots, x_{n+1})$. Formally

$$\mathbb{R}\mathrm{P}^n = (\mathbb{R}^{n+1} - \{0\}) / \sim,$$

where $x \sim y \Leftrightarrow (\exists \alpha \in \mathbb{R} - \{0\})(x = \alpha y)$. We define for $i = 1, \dots, n+1$ the set $U_i = \{[v] \in \mathbb{R}\mathrm{P}^n \mid x_i \neq 0\}$. Clearly $\mathbb{R}\mathrm{P}^n$ is covered by $U_1, \dots, U_{n+1}$. For each $[v] \in U_i$, we can choose a unique $v_1 \sim v$ such that the ith component of v_1 is 1 and then U_i is in one-to-one correspondence with the hyperplane $\{x \in \mathbb{R}^{n+1} \mid x_i = 1\}$, which can be projected onto $\mathbb{R}^n$ via the map p_i given by $(x_1, \dots, x_i, \dots, x_{n+1}) \mapsto (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{n+1})$. This is therefore a coordinate chart $\varphi_i \colon U_i \to \mathbb{R}^n$ and the set of all these charts is an atlas Γ on $\mathbb{R}\mathrm{P}^n$. It is rather easy to show that each function φ_i, φ_i^{-1} is computable with respect to δ_Γ and δ^n. We conclude that Γ is a computable atlas on $\mathbb{R}\mathrm{P}^n$.

The set $\mathcal{B}_\Gamma = \{\varphi_i^{-1}[\mu^n(w)] \mid i = 1, \dots, n+1; w \in \mathrm{dom}(\mu^n)\}$ is a base for the topology induced by Γ and also the property

$$\varphi_i^{-1}[\mu^n(w_1)] \subseteq \varphi_j^{-1}[\mu^n(w_2)] \cap \varphi_k^{-1}[\mu^n(w_3)]$$

is equivalent to

$$(\forall x \in \mathbb{R}^{n+1})(x_i \neq 0 \wedge p_i(x) \in \mu^n(w_1) \Rightarrow x_j \neq 0 \wedge p_j(x) \in \mu^n(w_2) \wedge x_k \neq 0 \wedge p_k(x) \in \mu^n(w_3)),$$

and using Theorem 1.2.6, the latter expression can be checked algorithmically. The effective space $\mathbf{RP}^n_\Gamma = (\mathbb{R}\mathrm{P}^n, \tau_\Gamma, \mathcal{B}_\Gamma, \lambda_\Gamma)$ is a computable topological space, which is equivalent to the computable space $\mathbf{T}_\Gamma(\mathbb{R}\mathrm{P}^n)$ induced by Γ.

All the previous examples are cases in which we can replace the canonical computable space $\mathbf{T}_\Phi(X)$ with the somewhat simpler effective space $\mathbf{X}_\Phi$, but according to Lemma 2.2.17, this can be done only when the latter is computable.

As with standard topological manifolds, it can happen that a set X has more that one computable atlas defined on it. We would like to consider two computable atlases that define the same computable topology as equivalent.

Definition 3.1.11. Two n-dimensional computable atlases $\Phi = \{(\varphi_i, U_i)\}, \Psi = \{(\psi_j, V_j)\}$ on X are *computably compatible* if and only if Φ and Ψ are topologically compatible (Definition 1.1.11) and $\delta_\Phi \sim \delta_\Psi$.

Computable compatibility of computable atlases is an equivalence relation on the set of all computable atlases on X.

Proposition 3.1.12. *Let $\Phi = \{(\varphi_i, U_i)\}, \Psi = \{(\psi_j, V_j)\}$ be two computable atlases on X. Then Φ and Ψ are computably compatible if and only if $\mathbf{T}_\Phi(X)$ and $\mathbf{T}_\Psi(X)$ are equivalent computable topological spaces.*

Proof. ($\Rightarrow$) That Φ and Ψ are computably compatible means that the following is true:

(1) Φ and Ψ are topologically compatible;

(2) $\delta_\Phi \sim \delta_\Psi$.

By (1) we have that $\tau = \tau_\Phi = \tau_\Psi$, thus $\mathbf{T}_\Phi(X) = (X, \tau, \beta_\Phi, \nu_\Phi)$ and $\mathbf{T}_\Psi(X) = (X, \tau, \beta_\Psi, \nu_\Psi)$. Combining (2) with part 2. of Lemma 2.1.16 and Theorem 2.2.13, $\mathbf{T}_\Phi(X)$ and $\mathbf{T}_\Psi(X)$ are equivalent.

($\Leftarrow$) If $\mathbf{T}_\Phi(X)$ and $\mathbf{T}_\Psi(X)$ are equivalent computable topological spaces, then $\tau_\Phi = \tau_\Psi$, thus Φ and Ψ induce the same topology on X, so that by Lemma 1.1.12 Φ and Ψ are topologically compatible. Also, by Theorem 2.2.13, $\delta_\Phi \sim \delta_\Psi$. Therefore, by Definition 3.1.11, Φ and Ψ are computably compatible. □

Example 3.1.13. There is an example [BC70] of two atlases Φ, Φ' with the same underlying set E, which are not topologically compatible. We describe E, Φ and Φ' now and we will also show that Φ and Φ' are computable atlases on E, but they are not computably compatible. Let $E \subset \mathbb{R}^2$ be the set $\{(\sin 2s, \sin s) \mid s \in \mathbb{R}\}$. E is known as the *Figure eight*. The injection $f_1 \colon E \to \mathbb{R}$ defined by

$$f_1(\sin 2s, \sin s) = s, \quad s \in (0, 2\pi)$$

is onto a θ^1-computable open interval of $\mathbb{R}$ (namely, the set $(0, 2\pi)$) and so it is a chart whose domain is the whole set E. We claim that f_1 is (δ_Φ, δ^1)-computable and its inverse f_1^{-1} is (δ^1, δ_Φ)-computable. Let $\mathfrak{M}$ be a Type-2 machine with the following program: On the input $p \in \operatorname{dom} \delta_\Phi$, $\mathfrak{M}$ enumerates all strings $\langle 1, w \rangle \ll p$ and for each such string, $\mathfrak{M}$ discards the symbol "1" and then it outputs $\iota(w)$ on the output tape, and then it continues with the next string inside p. The function computed by $\mathfrak{M}$ realizes f_1 with respect to δ_Φ and δ^1, thus f_1 is (δ_Φ, δ^1)-computable. To prove that the inverse f_1^{-1} is (δ^1, δ_Φ)-computable, we define a Type-2 machine $\mathfrak{N}$ with the following program. On the input $q \in \operatorname{dom} \delta^1$ with $\delta^1(q) = s \in (0, 2\pi)$:

```
1. For each string $w \ll q$, such that $s \in \mu^1(w)$,
    1.1 For each $n \in \mathbb{N}$,
        1.1.1 Compute $q_n \in \mathbb{Q}$ such that $d(q_n, 2\pi) < 2^{-n}$;
        1.1.2 if $\mu^1(w) \subseteq (0, q_n - 2^{-n})$, then output $\iota(\langle 1, w \rangle)$ and go back to step 1;
        1.1.3 Otherwise, if $0 \in \mu^1(w)$ or $(q_n - 2^{-n}, q_n + 2^{-n}) \subseteq \mu^1(w)$, then go back to step 1;
        1.1.4 Continue with next iteration in step 1.1.
```

Let $A = (0, 2\pi)$. For each string $w \ll q$, the main task of $\mathfrak{N}$ is to find out if $\mu^1(w) \subseteq A$ and when that occurs, output the string $\iota(\langle 1, w \rangle)$, where $\langle 1, w \rangle \in \operatorname{dom} \lambda_\Phi$ on the output tape. To determine if the open interval $\mu^1(w)$ is a subset of A, $\mathfrak{N}$ computes the rational q_n, which is a 2^{-n}-approximation[4] of 2π, that is, $2\pi \in B(q_n, 2^{-n}) = (q_n - 2^{-n}, q_n + 2^{-n})$. Then $\mathfrak{N}$ performs the test $\mu^1(w) \subseteq (0, q_n - 2^{-n})$; if this is successful, then it is true that $\mu^1(w) \subset A$, because $(0, q_n - 2^{-n}) \subset A$ and in this case, $\lambda_\Phi(\langle 1, w \rangle) = f_1^{-1}[\mu^1(w)]$ is defined and contains the point $f_1^{-1}(s)$, thus the output of $\mathfrak{N}$ in step 1.1.2 is correct. But if $\mu^1(w) \nsubseteq (0, q_n - 2^{-n})$, then $\mathfrak{N}$ tries to find out if $\mu^1(w) \nsubseteq A$ with the if's test in step 1.1.3. If $0 \in \mu^1(w)$, then it must exists $r \in \mu^1(w)$ such that $r < 0$ and that would imply that $r \notin A$. On the other hand, if $B(q_n, 2^{-n}) \subseteq \mu^1(w)$, then there is some $r' \in \mu^1(w)$ with $2\pi < r'$, thus $r' \notin A$. Either the if's test in step 1.1.2 is successful or the test in step 1.1.3 is successful, the behavior of $\mathfrak{N}$ is correct with respect to the string w and then $\mathfrak{N}$ goes back to step 1 to process the next string in q.

[4]See the proof of Lemma 2.1.10.

If none of the test in steps 1.1.2 and 1.1.3 are successful with the rational q_n, then $\mathfrak{N}$ cannot determine yet what to do with the string w and it must go back to step 1.1 to compute a better approximation of 2π to execute steps 1.1.2-1.1.4. Given $w \ll q$, $\mathfrak{N}$ only needs to compute a finite number of approximations of 2π, because it is true that

$$\mu^1(w) \subseteq A \Leftrightarrow (\exists n_0)(\mu^1(w) \subseteq (0, q_{n_0} - 2^{-n_0}));$$

$$\mu^1(w) \nsubseteq A \Leftrightarrow 0 \in \mu^1(w) \text{ or } (\exists n_0)((q_{n_0} - 2^{-n_0}, q_{n_0} + 2^{-n_0}) \subseteq \mu^1(w)).$$

Therefore the output of $\mathfrak{N}$ is correct for every w, thus the function computed by $\mathfrak{N}$ realizes f_1^{-1} with respect to δ^1 and δ_Φ.

To prove that every step of $\mathfrak{N}$ can be done in finite time and in an algorithmic way, we only need to prove that steps 1.1.1-1.1.4 are computable. Step 1.1.1 can be done by $\mathfrak{N}$ because by Lemma 2.1.10, π is δ^1-computable, so that 2π is δ^1-computable too. Steps 1.1.3 and 1.1.4 involve some set operation with open sets defined in terms of rational numbers. These operations can be done using some inequality between rational numbers and the string $\iota(\langle 1, w\rangle)$ is clearly computable. We conclude that all the steps of $\mathfrak{N}$ are computable in finite time.

Thus the function f_1 and its inverse are computable with respect to δ_Φ and δ^1. Therefore Φ is a 1-dimensional computable atlas of E. Another computable atlas Φ' of E consisting of a single chart can be constructed by using the function $g\colon E \to \mathbb{R}$ defined by

$$g(\sin 2s, \sin s) = s, \quad s \in (-\pi, \pi).$$

The proof of the computability of g and its inverse with respect to $\delta_{\Phi'}$ and δ^1 is almost identical to that of the computability of f_1 and its inverse. So that Φ and Φ' are two computable atlases on the set E. But these two atlases of E are not compatible, since the change of coordinates $g \circ f_1^{-1}\colon f_1[E] \to g[E]$ is given by

$$g \circ f_1^{-1}(s) = \begin{cases} s & \text{if } 0 < s < \pi \\ 0 & \text{if } s = \pi \\ s - 2\pi & \text{if } \pi < s < 2\pi \end{cases} \tag{3.4}$$

and this function is not even continuous, thus Φ and Φ' are not topologically compatible, so that they cannot be computably compatible.

The fact that the two atlases of example 3.1.13 are not computably compatible comes from the fact that these atlases are not topologically compatible. We now show two computable atlases which are compatible, but not computably compatible.

Example 3.1.14. Let $\Phi = \{(1_\mathbb{R}, \mathbb{R})\}$ and $\Psi = \{(\varphi_1, \mathbb{R})\}$, where $\varphi_1\colon \mathbb{R} \to \mathbb{R}$ is defined as $\varphi_1(x) = x + a$. Ψ is the atlas of Example 3.1.4, where we saw that the computability of Ψ does not depend on the computability of the number a with respect to δ^1. We now prove that if a is not δ^1-computable, then Φ and Ψ are not computably compatible (notice that the two atlases are topologically compatible).

Suppose then that a is not computable with respect to δ^1 and that Φ and Ψ are computably compatible. By Definition 3.1.11, $\delta_\Phi \sim \delta_\Psi$ and as Φ is the atlas induced by the identity on $\mathbb{R}$, it is true that $\delta_\Phi \sim \delta^1$, thus we have that $\delta_\Psi \sim \delta^1$. Because Ψ is computable, we have that φ_1 is (δ_Ψ, δ^1)-computable. Since $\delta_\Psi \sim \delta^1$, we can conclude that φ_1 is (δ^1, δ^1)-computable. But φ_1 is computable with respect to δ^1 if and only if a is a computable real number. Therefore Φ and Ψ cannot be computably compatible.

Example 3.1.14 give us a desirable consequence of our definitions. If the map φ_1 is not computable in $\mathbb{R}$ with the usual manifold structure, then we do not want the atlas Ψ to be compatible with Φ.

Definition 3.1.15. A *computable structure* on X is an equivalence class $[\Phi]$ of computable atlases on X.

Definition 3.1.16 (Computable manifold)**.** An n-dimensional *computable manifold* is a set M together with a computable structure $[\Phi]$.

Thus, a computable n-manifold is a pair $(M, [\Phi])$. As each computable atlas determines a unique computable structure, we can also simply write (M, Φ) and forget about the brackets and sometimes, if no confusion arises, we will omit the explicit reference to the computable atlas Φ. The integer n, the dimension of the manifold, is denoted in the usual form as $\dim M$. All the previous examples about sets with computable atlases are actually computable manifolds, we now give more examples.

Example 3.1.17. We show that for $n \geqslant 1$, the n-disk $D^n \subset \mathbb{R}^n$ is a computable n-manifold with boundary. A topological atlas for D^n can be build as follows. Let U be the topological interior of D^n ($U = B(0,1)$) and $f\colon U \to \mathbb{R}^n_+$ be the map $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_n + 1)$, this is clearly a bijective map from U onto a θ^n_+-computable open set of $\mathbb{R}^n_+$. Let $r \in \{1, -1\}$, $P_r = (0, \ldots, r)$ and $V_r = D^n - \{P_r\}$. Define $g_r\colon V_r \to \mathbb{R}^n$ as

$$g_r(x) = P_r + \frac{x - P_r}{\|x - P_r\|^2}.$$

The map g_r sends V_r onto a subspace of $\mathbb{R}^n$, homeomorphic to $\mathbb{R}^n_+$ and we can use the g_r's to define charts $f_r\colon V_r \to \mathbb{R}^n_+$ as follows:

$$f_1(x) = g_1(x) + P_1;$$

$$f_{-1}(x) = r(g_{-1}(x) + P_{-1});$$

where $r\colon \mathbb{R}^n \to \mathbb{R}^n$ is given by $r(x_1, \ldots, x_n) = (x_1, \ldots, -x_n)$. Notice that $f_r[V_r] = \mathbb{R}^n_+$, thus the image of each f_r is θ^n_+-computable in $\mathbb{R}^n_+$. Using f, f_1 and f_{-1}, we obtain an atlas $\Psi = \{(f, U), (f_1, V_1), (f_{-1}, V_{-1})\}$ on D^n. To prove that Ψ is computable, we can use the same technique that we used in Example 3.1.9, i.e., we first show that $\mathbf{T}_\Psi(D^n)$ and $D^n_{\mathbb{R}^n}$ are equivalent computable topological spaces with the aid of Theorem 1.2.6; second we prove that f, f_1, f_{-1} and their inverses are computable with respect to $\delta^n_{D^n}$ and δ^n_+, by using Lemma 2.2.5. Thus, the charts of Ψ are $(\delta_\Psi, \delta^n_+)$-computables and their inverses are $(\delta^n_+, \delta_\Psi)$-computables. The pair (D^n, Ψ) is a computable n-manifold with boundary. We know from standard manifold theory that $\operatorname{Int} D^n = B(0,1)$ and $\partial D^n = \mathbb{S}^{n-1}$.

Example 3.1.18. A computable 0-manifold is just a discrete computable topological space.

Proposition 3.1.19. *Let $(M_1, \Phi_1), (M_2, \Phi_2)$ be computable manifolds such that* $\dim M = n$ *and* $\dim N = m$*. Then there exists a $(n + m)$-dimensional computable atlas $\overline{\Phi}$ for the set $M_1 \times M_2$ with the following properties:*

(a) $\lambda_{\overline{\Phi}} = \lambda_{\Phi_1} \times \lambda_{\Phi_2}$;

(b) $\delta_{\overline{\Phi}} \sim [\delta_{\Phi_1}, \delta_{\Phi_2}]$*;*

(c) $\mathbf{T}_{\overline{\Phi}}(M_1 \times M_2) = \overline{\mathbf{T}}$*, where* $\overline{\mathbf{T}}$ *is the computable topological space of Lemma 2.1.20 induced by the computable spaces* $\mathbf{T}_{\Phi_1}(M_1)$ *and* $\mathbf{T}_{\Phi_2}(M_2)$*.*

Proof. Let $\overline{\mathbf{X}} = (\mathbb{R}^n_+ \times \mathbb{R}^m_+, \overline{\tau}, \overline{\beta}, \overline{\nu})$ be the computable product space induced by $\mathbf{R}^n_+$ and $\mathbf{R}^m_+$. Then by Lemma 2.2.11, there exists a computable homeomorphism $g\colon \mathbb{R}^n_+ \times \mathbb{R}^m_+ \to \mathbb{R}^{n+m}_+$ and with the map g, the set $M_1 \times M_2$ endowed with the atlas

$$\overline{\Phi} = \{(g \circ (\varphi \times \psi), U \times V) \mid (\varphi, U) \in \Phi_1, (\psi, V) \in \Phi_2\}$$

is a topological manifold of dimension $n + m$, such that the topology induced in $M_1 \times M_2$ by $\overline{\Phi}$ is the product topology of the spaces M_1 and M_2. We show that $\overline{\Phi}$ is computable and we prove (a)-(c). Notice that since $\overline{\mathbf{X}}$ and $\mathbf{R}^{n+m}_+$ are computably homeomorphic computable spaces, in $\mathbf{R}^{n+m}_+$ we can replace the base β^{n+m}_+ by the base

$$\overline{\beta}_g = \{g[\mu^n_+(w) \times \mu^m_+(z)] \mid (w, z) \in \mathrm{dom}(\mu^n_+) \times \mathrm{dom}(\mu^m_+)\},$$

(see Lemma 2.2.15 and Corollary 2.2.16), thus the elements of the computable predicate space $\mathbf{Z}_{\overline{\Phi}} = (M_1 \times M_2, \mathcal{B}_{\overline{\Phi}}, \lambda_{\overline{\Phi}})$ are defined as

$$\mathcal{B}_{\overline{\Phi}} = \{(g \circ (\varphi_i \times \psi_j))^{-1}[B] \mid B = g[\mu^n_+(w) \times \mu^m_+(z)] \wedge (\langle i, w\rangle, \langle j, z\rangle) \in \mathrm{dom}\, \nu_{\Phi_1} \times \mathrm{dom}\, \nu_{\Phi_2}\};$$

$$\lambda_{\overline{\Phi}}\colon \Sigma^* \to \mathcal{B}_{\overline{\Phi}}, \lambda_{\overline{\Phi}}(\langle\langle i, j\rangle, \langle w, z\rangle\rangle) = (g \circ (\varphi_i \times \psi_j))^{-1}[g[\mu^n_+(w) \times \mu^m_+(z)]].$$

Hence, we have that $\lambda_{\overline{\Phi}}(\langle\langle i, j\rangle, \langle w, z\rangle\rangle) = (g \circ (\varphi_i \times \psi_j))^{-1}[g[\mu^n_+(w) \times \mu^m_+(z)]] = (\varphi_i^{-1} \times \psi_j^{-1}) \circ g^{-1}[g[\mu^n_+(w) \times \mu^m_+(z)]] = (\varphi_i^{-1} \times \psi_j^{-1})[\mu^n_+(w) \times \mu^m_+(z)] = \varphi_i^{-1}[\mu^n_+(w)] \times \psi_j^{-1}[\mu^m_+(z)] = \lambda_{\Phi_1}(\langle i, w\rangle) \times \lambda_{\Phi_2}(\langle j, z\rangle) = (\lambda_{\Phi_1} \times \lambda_{\Phi_2})(\langle i, w\rangle, \langle j, z\rangle)$ and this implies that $\delta_{\overline{\Phi}} \sim [\delta_{\Phi_1}, \delta_{\Phi_2}]$; this proves (a) and (b). It is easy to verify that the computable topological space $\mathbf{T}_{\overline{\Phi}}(M_1 \times M_2) = (M_1 \times M_2, \tau_{\overline{\Phi}}, \beta_{\overline{\Phi}}, \nu_{\overline{\Phi}})$ induced in $M_1 \times M_2$ by $\overline{\Phi}$ is precisely the computable product space of $\mathbf{T}_{\Phi_1}(M_1)$ and $\mathbf{T}_{\Phi_2}(M_2)$, so that (c) is true.

To finish the proof of the Lemma, we only need to show that $\overline{\Phi}$ is a computable atlas on $M_1 \times M_2$, we have to prove that each chart $g \circ (\varphi_i \times \psi_j)\colon U_i \times V_j \to \mathbb{R}^{n+m}_+ \in \overline{\Phi}$ is $(\delta_{\overline{\Phi}}, \delta^{n+m}_+)$-computable with $(\delta^{n+m}_+, \delta_{\overline{\Phi}})$-computable inverse and that each set $\varphi_i[U_i] \times \psi_j[V_j]$ is θ^{n+m}_+-computable. To do this, we will use the computable space $\overline{\mathbf{X}}$ and the representations $\overline{\delta}, \overline{\theta}$ of points and open sets of $\mathbb{R}^n_+ \times \mathbb{R}^m_+$ induced by $\overline{\mathbf{X}}$. By (b), $\delta_{\overline{\Phi}} \sim [\delta_{\Phi_1}, \delta_{\Phi_2}]$ and by part 2. of Lemma 2.1.20, $\varphi_i \times \psi_j$ is $(\delta_{\overline{\Phi}}, \overline{\delta})$-computable with $(\overline{\delta}, \delta_{\overline{\Phi}})$-computable inverse. Using part 3. of Lemma 2.1.20, the set $\varphi_i[U_i] \times \psi_j[V_j]$ is a $\overline{\theta}$-computable open subset of $\mathbb{R}^n_+ \times \mathbb{R}^m_+$. Combining these facts with the computable homeomorphism g between $\overline{\mathbf{X}}$ and the computable half space $\mathbf{R}^{n+m}_+$, we deduce that $\overline{\Phi}$ is a computable atlas on $M_1 \times M_2$. □

We can generalize this result to arbitrary finite products of computable manifolds $M_1, \ldots, M_r$.

Example 3.1.20. Following Proposition 3.1.19, we can prove that the *n-dimensional torus*

$$T^n = \prod_{i=1}^{n} \mathbb{S}^1$$

is a computable n-manifold, where we equip $\mathbb{S}^1$ with any of the atlases given in Examples 3.1.7 and 3.1.9 (they are computably compatible).

3.2 Morphisms between computable manifolds

In the case of general topological manifolds, relationships between two given manifolds can be studied by means of continuous functions. In this section, we will show that (as expected) computable functions, as we have defined them in Section 2.2, are just adequate to be used as morphisms between computable manifolds.

3.2.1 Computable functions

To define morphisms between two computable manifolds (M, Φ) and (N, Ψ), we have two possible choices. The first uses the induced computable topological spaces $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$, a morphism between M and N is just a computable continuous function between $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$. The second approach uses the representations δ_Φ, δ_Ψ of M and N respectively, induced by the computable predicate spaces $\mathbf{Z}_\Phi$ and $\mathbf{Z}_\Psi$. The two options are equivalent thanks to Definition 2.2.4 and part 2. of Lemma 2.1.16. As our formal definition, we adopt the second approach.

Definition 3.2.1. Let $(M, \Phi), (N, \Psi)$ be computable manifolds of dimensions n and m respectively. A morphism $f\colon \subseteq (M, \Phi) \to (N, \Psi)$ between computable manifolds is a $(\delta_\Phi, \delta_\Psi)$-computable function $f\colon \subseteq M \to N$, where δ_Φ, δ_Ψ are the representations induced by $\mathbf{Z}_\Phi$ and $\mathbf{Z}_\Psi$ respectively.

Lemma 3.2.2. *Let $(M, \Phi), (N, \Psi)$ be computable manifolds and $f\colon \subseteq M \to N$ a function. Then f is a morphism of computable manifolds if and only if f is a computable map between the computable spaces $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$.*

Proof. $(\Rightarrow)$ If f is a morphism of computable manifolds, then f is $(\delta_\Phi, \delta_\Psi)$-computable. But we know from part 2 of Lemma 2.1.16 that δ_Φ is equivalent to the inner representation of points of the computable space $\mathbf{T}_\Phi(M) = T(\mathbf{Z}_\Phi)$. A similar statement is true for δ_Ψ and $\mathbf{T}_\Psi(N)$. Therefore f is a computable function with respect to the inner representations of $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$, so that by Definition 2.2.4, f is a computable map between $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$.

$(\Leftarrow)$ If f is a computable map between $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$, then by Definition 2.2.4, f is computable with respect to the inner representations of points of $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(N)$ and these representations are equivalent to the representations δ_Φ and δ_Ψ respectively (by 2 of Lemma 2.1.16), so that by Definition 3.2.1, f is a morphism between the computable manifolds M and N. □

Remember that computable function are continuous. Of course, we would like computable homeomorphisms to be the standard equivalence between computable manifolds. We now prove some results about this topic. First, we have an analog of Lemma 2.2.15 for computable manifolds.

Lemma 3.2.3. *Let $(M, [\Phi])$ be a computable n-manifold and X a set. If $f\colon M \to X$ is a bijective function, then there exists a computable structure $[\Phi_f]$ on X induced by f and Φ, such that for all $z \in \operatorname{dom} \lambda_{\Phi_f}, \lambda_{\Phi_f}(z) = f[\lambda_\Phi(z)]$ and $\delta_{\Phi_f} = f \circ \delta_\Phi$.*

Proof. Let $\Phi = \{(\varphi_i, U_i)\}_{i \in I}$. By Lemma 2.2.15, $\mathbf{X}_f$ becomes a computable topological space and f is a computable homeomorphism between $\mathbf{T}_\Phi(M)$ and $\mathbf{X}_f$. Now, the atlas induced by f on X is given by

$$\Phi_f = \{(\psi_i, V_i) \mid \psi_i = \varphi_i \circ f^{-1} \text{ and } V_i = f[U_i], (\varphi_i, U_i) \in \Phi\}_{i \in I},$$

which induces the computable predicate space $\mathbf{Z}_{\Phi_f} = (X, \mathcal{B}_{\Phi_f}, \lambda_{\Phi_f})$, where

$\mathcal{B}_{\Phi_f} = \{(\varphi_i \circ f^{-1})^{-1}[\mu^n(w)] \mid (i,w) \in \Sigma^* \times \mathrm{dom}(\mu^n_+)\}$,

$\lambda_{\Phi_f}\colon \subseteq \Sigma^* \to \mathcal{B}_{\Phi_f}, \lambda_{\Phi_f}(\langle i,w\rangle) = (\varphi_i \circ f^{-1})^{-1}[\mu^n_+(w)]$,

and clearly $\lambda_{\Phi_f}(j) = f[\lambda_\Phi(j)]$, so that $\delta_{\Phi_f}(p) = f \circ \delta_\Phi(p)$ for all $p \in \mathrm{dom}\,\delta_{\Phi_f}$. For each $i \in I$, the open set $\psi_i[V_i] = (\varphi_i \circ f^{-1})[f[U_i]] = \varphi_i[U_i]$ is θ^n_+-computable open in $\mathbb{R}^n_+$ and the map ψ_i is $(\delta_{\Phi_f}, \delta^n_+)$-computable with $(\delta^n_+, \delta_{\Phi_f})$-computable inverse. It follows that Φ_f satisfies Definition 3.1.2, thus it induces a computable structure $[\Phi_f]$ on X. □

Corollary 3.2.4. *With the hypothesis of Lemma 3.2.3, $\mathbf{T}_{\Phi_f}(X) = \mathbf{X}_f$, where $\mathbf{X}_f$ is the computable topological space of Lemma 2.2.15.*

Proof. The topology of $\mathbf{T}_{\Phi_f}(X)$ is generated by the base

$$\beta_{\Phi_f} = \{f[B] \mid B \in \beta_\Phi\},$$

and this is exactly the set β_f; and we have that $\nu_{\Phi_f} = \nu_f$. In other words, $\mathbf{T}_{\Phi_f}(X) = \mathbf{X}_f$. □

Corollary 3.2.5. *Let (M,Φ) be a computable manifold and let $\mathbf{X} = (X,\tau,\beta,\nu)$ be a computable topological space. If $f\colon M \to X$ is a computable homeomorphism between $\mathbf{T}_\Phi(M)$ and $\mathbf{X}$, then $\mathbf{T}_{\Phi_f}(X)$ is equivalent to $\mathbf{X}$.*

Proof. An immediate consequence of Lemma 2.2.16 and Corollary 3.2.4. □

Corollary 3.2.6. *Let $(M,\Phi), (N,\Phi')$ be computable manifolds. If $f\colon M \to N$ is a computable homeomorphism, then the computable structure of N and the computable structure induced by f on N are the same.*

Proof. By Corollary 3.2.5, $\mathbf{T}_{\Phi_f}(N)$ and $\mathbf{T}_{\Phi'}(N)$ are equivalent computable topological spaces, so that we can use Proposition 3.1.12 to conclude that Φ_f and Φ' are computably compatible (Definition 3.1.11), therefore $[\Phi_f] = [\Phi']$. □

With this last result, we can see that two computably homeomorphic computable manifolds are "essentially" the same manifold, just in the same way that happens with homeomorphic topological manifold.

Example 3.2.7. Let $h = s^{-1}\colon \mathbb{R}^n \to S_P$ ($S_P = \mathbb{S}^n - \{P\}$) be the inverse of the stereographic projection of Example 2.2.10. If we give $\mathbb{R}^n$ with the (canonical) manifold structure of Example 3.1.3, then by Lemma 3.2.3 and Corollary 3.2.5, S_P becomes a computable n-manifold with a computable structure Γ such that $\mathbf{T}_\Gamma(S_P)$ is equivalent to $Q_{\mathbf{S}^n}$, where $Q = S_P$.

Example 3.2.8. In example 3.1.13, we showed two non-compatible computable atlases $\Phi = \{(f,E)\}$, $\Phi' = \{(g,E)\}$ on the set $E = \{(\sin 2s, \sin s) \mid s \in \mathbb{R}\}$ known as the Figure eight. The function $h\colon E \to E$ of the the set E equipped with the computable topology induced by Φ, onto the same set using the computable topology induced by Φ' given by

$$h(\sin 2s, \sin s) = (\sin(2(s-\pi)), \sin(s-\pi))$$

is a computable homeomorphism from $\mathbf{T}_\Phi(E)$ onto $\mathbf{T}_{\Phi'}(E)$. So that although Φ, Φ' are not computably compatible, the two manifolds (E,Φ) and (E,Φ') are computably homeomorphic, that is, they are essentially the same manifold.

Example 3.2.9. In Example 3.1.14, we proved for the real line $\mathbb{R}$, the atlases $\Phi = \{(1_{\mathbb{R}}, \mathbb{R})\}$ and $\Psi = \{(\varphi_1, \mathbb{R})\}$ are not computably compatible, if the map $\varphi_1 \colon \mathbb{R} \to \mathbb{R}$ defined as $\varphi_1(x) = x + a$ is not (δ^1, δ^1)-computable, which is equivalent to non-computability of the number a. However, the two manifolds $(\mathbb{R}, \Phi)$ and $(\mathbb{R}, \Psi)$ can be seen to be computably homeomorphic, as the map φ_1 is (δ_Ψ, δ^1)-computable and its inverse is (δ^1, δ_Ψ)-computable (Example 3.1.4), thus it is a computable homeomorphism. This tell us that although the atlases Φ and Ψ are not computably compatible (whenever a is not δ^1-computable), we can transform one manifold into the other in a computable way and that means that everything that is computable in one manifold, is computable in the other.

We will have more to say about computable functions between computable manifolds in Chapter 4, where we introduce *computable submanifolds* and study *computable embeddings* of manifolds.

3.3 Properties of computable manifolds

We now present some properties of computable manifolds. First, we analyze which topological properties are satisfied by the induced topology on a computable manifold M, then we present some results about computability in M. Finally, we construct the computable versions of some classical results about special atlases on M.

3.3.1 Topological and computable properties

By definition, a computable topological space is a T_0-space and thus, every computable manifold is a T_0-space. However, a manifold also satisfies all the local topological properties of euclidean space $\mathbb{R}^n$. Here is a list of some basic properties that the induced topology on a computable manifold fulfills. The induced topology on a computable manifold satisfies:

- The T_1 separation axiom;
- the second axiom of countability (because every computable manifold has a countable atlas);
- it is locally connected;
- it is locally compact.

See [Gau74] for details and even more topological properties that a computable manifold satisfies. But despite the fact that $\mathbb{R}^n$ is a (computably) Hausdorff space, it is not true that every computable manifold is (computably) Hausdorff. We now prove that a well known example of a non-Hausdorff topological manifold is in fact, an example of a non-Hausdorff computable manifold.

Example 3.3.1. Let $H \subset \mathbb{R}^2$ be defined by $H = \{(s,0) \mid s \in \mathbb{R}\} \cup \{(0,1)\}$. H is called *the line with two origins*. Let U be the subset of H of all points of the form $(s,0)$ with $s \in \mathbb{R}$ and $U' = (U - \{(0,0)\}) \cup \{(0,1)\}$. Define charts $f \colon U \to \mathbb{R}$ and $f' \colon U' \to \mathbb{R}$ of H into $\mathbb{R}$ by

$$f(s,0) = s$$

$$f'(s,0) = s \text{ for } s \neq 0 \text{ and } f'(0,1) = 0$$

It is very easy to show that $\Phi = \{(f,U),(f',U')\}$ is an atlas on H and that each chart is (δ_Φ, δ^1)-computable with (δ^1, δ_Φ)-computable inverse. Since $f[U] = f'[U'] = \mathbb{R}$, each of these sets is θ^1-computable open in the computable space $\mathbf{R}$. The atlas Φ satisfies Definition 3.1.2 so that the pair (H, Φ) becomes a computable 1-manifold. The proof that H is a non-Hausdorff space with the topology induced by Φ can be found in [Gau74, BC70].

Recall from Section 3.1.1 that given a computable n-manifold (M, Φ) using the atlas Φ, we constructed the computable predicate space $\mathbf{Z}_\Phi = (M, \mathcal{B}_\Phi, \lambda_\Phi)$ where $\mathcal{B}_\Phi$ is the set of all computable balls in M. This predicate space depends on Φ and it can happen that $\mathcal{B}_\Phi$ contains empty elements. If the subset $E \subseteq \operatorname{dom} \lambda_\Phi$ defined by

$$E = \{l \in \operatorname{dom} \lambda_\Phi \mid \lambda_\Phi(l) \neq \varnothing\} \tag{3.5}$$

is c.e., then (M, Φ) will be called a *computable manifold with non-empty computable balls*. The next lemma states that if the set of non-empty computable balls is c.e., then the empty elements of $\mathcal{B}_\Phi$ can be removed from the computable spaces $\mathbf{Z}_\Phi$ and $\mathbf{T}_\Phi(M)$[5].

Lemma 3.3.2. *Let (M, Φ) be a computable n-manifold with non-empty computable balls. Then there exists a computable topological space $\mathbf{T}'(M) = (M, \tau_\Phi, \beta', \nu')$ such that $\mathbf{T}_\Phi(M)$ is equivalent to $\mathbf{T}'(M)$ and $\beta' \subset \beta_\Phi$ is the set of all finite intersections of non-empty computable balls of M.*

Proof. Let $\mathcal{E} = \lambda_\Phi(E)$. Because the set $E \subseteq \operatorname{dom} \lambda_\Phi$ is c.e. (and infinite), we can apply Lemma 1.3.9 to obtain an injective total computable function $h\colon \Sigma^* \to \Sigma^*$ such that range $h = E$. Define the notation $\lambda_h\colon \Sigma^* \to \mathcal{E}$ by $\lambda_h(w) = \lambda_\Phi \circ h(w)$. Then the triple $\mathbf{Z}_h = (M, \mathcal{E}, \lambda_h)$ is a computable predicate space and as $\lambda_h = \lambda_\Phi \circ h$ and $\lambda_\Phi = \lambda_h \circ h^{-1}$, we have that $\lambda_\Phi \sim \lambda_h$, and this fact implies that $\delta_\Phi \sim \delta_h = \delta_{\mathbf{Z}_h}$. The computable topological space induced by $\mathbf{Z}_h$ is

$$\mathbf{T}'(M) = (M, \tau_\Phi, \beta', \nu'),$$

where β' is the base generated by all finite intersections of the elements of $\mathcal{E} \subseteq \mathcal{B}_\Phi$, thus $\beta' \subset \beta_\Phi$. Since $\delta \sim \delta_\Phi \sim \delta_h \sim \delta'$, where δ, δ' are the inner representations of M induced by $\mathbf{T}_\Phi(M)$ and $\mathbf{T}'(M)$ respectively, we conclude that $\mathbf{T}_\Phi(M)$ and $\mathbf{T}'(M)$ are equivalent computable topological spaces, hence the result follows. $\square$

In a computable n-manifold, the computable points can be characterized by the computability of points in the computable euclidean space $\mathbf{R}^n$. We present a simple result from which other characterizations can be derived.

Lemma 3.3.3. *Let (M, Φ) be a computable n-manifold and $x \in M$. Then x is a δ_Φ-computable point in M if and only if there exists a δ^n-computable point $y \in \mathbb{R}^n$ and a computable function $f\colon \subseteq \mathbb{R}^n \to M$ such that $f(y) = x$.*

Proof. ($\Rightarrow$) Let $x \in M$ be a δ_Φ-computable point. There exists a chart $(\varphi, U) \in \Phi$ such that $x \in U$ and φ is a computable homeomorphism between U and an open subset of $\mathbb{R}^n_+$. In particular, φ and φ^{-1} are computable with respect to δ^n_+ and δ_Φ, which implies that φ and φ^{-1} preserve the

[5]This result is reminiscent of Lemma 25 of [WG09], which says that for a computable topological space $\mathbf{X}$, the empty base elements can be ignored if the set of non-empty base elements is c.e.

computability of points between U and $\varphi[U]$. Let $y = \varphi(x)$ and $f = \varphi^{-1}$. Then y is a δ^n_+-computable point in $\mathbb{R}^n_+$ (thus, y is a δ^n-computable point in $\mathbb{R}^n$) and f is a computable map such that $f(y) = x$.

($\Leftarrow$) Because f is computable, it takes the δ^n-computable point $y \in \mathbb{R}^n$ onto a δ_Φ-computable point in M, thus $f(y) = x$ must be δ_Φ-computable in M. □

3.3.2 Some special computable atlases

In Theorem 3.3.5, we deal with the existence of two very useful computable atlases. To show that one of these atlases exists, our computable manifolds will need the property of non-empty computable balls and a technical result about computable homeomorphisms between rational open balls $B(q, \epsilon) \subset \mathbb{R}^n$ and $\mathbb{R}^n$.

Lemma 3.3.4. *In the computable euclidean space $\mathbf{R}^n$, the following statements hold:*

(1) For each $w \in \operatorname{dom}(\mu^n)$ such that $\mu^n(w) = B_w = B(q, \epsilon)$, there exists a computable homeomorphism $h_w \colon B_w \to \mathbb{R}^n$ from $(B_w)_{\mathbf{R}^n}$ to $\mathbf{R}^n$.

(2) For each $z \in \operatorname{dom}(\mu^n_+)$ such that $\mu^n_+(z) \cap \partial\mathbb{R}^n_+ \neq \varnothing$, there exists a computable homeomorphism $g_z \colon \mu^n_+(z) \to \mathbb{R}^n_+$ from $\mu^n_+(z)_{\mathbf{R}^n}$ to $\mathbf{R}^n_+$.

(3) The sets

$$\{(v, w, z) \in (\operatorname{dom}(\mu^n))^3 \mid \mu^n(v) \subset h_w^{-1}[\mu^n(z)]\} \tag{3.6}$$

$$\{(s, w, z) \in (\operatorname{dom}(\mu^n_+))^3 \mid \mu^n_+(s) \subset g_w^{-1}[\mu^n_+(z)]\} \tag{3.7}$$

are computable, where h_w and g_w are the computable homeomorphisms of part (1) and (2) respectively.

Proof. (1) Let $h \colon B(0, 1) \to \mathbb{R}^n$ be the computable homeomorphism of Example 2.2.9 and let $S_\varepsilon, T_a \colon \mathbb{R}^n \to \mathbb{R}^n$ ($a \in \mathbb{R}^n, 0 < \varepsilon \in \mathbb{R}$) be homeomorphisms of $\mathbb{R}^n$ defined as

$$T_a(x) = x - a \quad \text{and} \quad S_\varepsilon(x) = \varepsilon x.$$

If $(a, \varepsilon) \in \mathbb{Q}^n \times \mathbb{Q}^+$, then T_a, S_ε are computable homeomorphisms of $\mathbf{R}^n$ onto itself. Now, if $\mu^n(w) = B(q, \varepsilon)$, then we can define $h_w \colon B(q, \varepsilon) \to \mathbb{R}^n$ as $h_w = h \circ S_{\varepsilon^{-1}} \circ T_q$, clearly h_w is a computable homeomorphism of B_w onto $\mathbb{R}^n$.

(2) We can define the homeomorphism g_z by $g_z = h_z|_{\mu^n_+(z)}$. It is easy to see that g_z is a computable homeomorphism between $\mu^n_+(z)$ and $\mathbb{R}^n_+$.

(3) Using the computable homeomorphisms h_w of (1), we can express the property $\mu^n(v) \subset h_w^{-1}[\mu^n(z)]$ in terms of polynomial functions in n variables and rational coefficients, this is true because we know that

$$\mu^n(v) \subset h_w^{-1}[\mu^n(z)] \Leftrightarrow (\forall x \in \mathbb{R}^n)(x \in \mu^n(v) \Rightarrow h_w(x) \in \mu^n(z)) \tag{3.8}$$

and it can be seen that the right side of (3.8) is equivalent to the expression

$$(\forall x \in \mathbb{R}^n)(d(x, q_v) < \varepsilon_v \Rightarrow d(h_w(x), q_z) < \varepsilon_z), \tag{3.9}$$

where $\mu^n(v) = B(q_v, \varepsilon_v)$ and $\mu^n(z) = B(q_z, \varepsilon_z)$. Now, using the formula to compute the function h_w, which is defined as

$$h_w(x) = \frac{\varepsilon^{-1}(x - q)}{1 - \|\varepsilon^{-1}(x - q)\|}, \quad (\mu^n(w) = B(q, \varepsilon))$$

for the distance $d(h_w(x), q_z)$, we have that

$$\begin{aligned} d(h_w(x), q_z) &= d\left(\frac{\varepsilon^{-1}(x-q)}{1-\|\varepsilon^{-1}(x-q)\|}, q_z\right) \\ &= \frac{1}{1-\|\varepsilon^{-1}(x-q)\|} d(\varepsilon^{-1}(x-q), (1-\|\varepsilon^{-1}(x-q)\|)q_z). \end{aligned}$$

Using the last expression, we can deduce that

$$d(h_w(x), q_z) < \varepsilon_z \Leftrightarrow d(\varepsilon^{-1}(x-q), (1-\|\varepsilon^{-1}(x-q)\|)q_z) < \varepsilon_z(1-\|\varepsilon^{-1}(x-q)\|). \quad \boxed{3.10}$$

Let $y = \varepsilon^{-1}(x-q)$. Now we can write Equation $\boxed{3.9}$ as

$$(\forall x, y \in \mathbb{R}^n)(y = \varepsilon^{-1}(x-q) \wedge d(x, q_v) < \varepsilon_v \Rightarrow d(y, (1-\|y\|)q_z) < \varepsilon_z(1-\|y\|)).$$

Now let $r = 1 - \|y\| \in \mathbb{R}$, then $\|y\|^2 = (1-r)^2$ and the above equation becomes

$$(\forall x, y \in \mathbb{R}^n, \forall r \in \mathbb{R})(y = \varepsilon^{-1}(x-q) \wedge \|y\|^2 = (1-r)^2 \wedge d(x, q_v) < \varepsilon_v \Rightarrow d(y, rq_z) < \varepsilon_z r). \quad \boxed{3.11}$$

This last expression can be easily converted into an elementary expression and it can be seen that given $v, w, z \in \operatorname{dom}(\mu^n)$ the polynomial expression defining Equation $\boxed{3.11}$ can be constructed algorithmically, so that the decidability of $\boxed{3.6}$ can be verified by a single Turing machine (using Theorem 1.2.6) uniformly in $(v, w, z) \in (\operatorname{dom}(\mu^n))^3$. To show that the set of Equation $\boxed{3.7}$ is computable, the argument is very much the same as for the set of Equation $\boxed{3.6}$, thus we omit it. □

Theorem 3.3.5 (Special computable atlases). *Let (M, Φ) be a computable n-manifold.*

(a) There exists a computable atlas Γ on M, computably compatible with Φ and such that for every chart $h\colon V \to \mathbb{R}^n_+ \in \Gamma$, $h[V] = B(q, \epsilon) \cap \mathbb{R}^n_+$ for some $(q, \epsilon) \in \mathbb{Q}^n \times \mathbb{Q}^+$.

(b) If (M, Φ) has the property of non-empty computable balls and $\partial M = \varnothing$, there exists a computable atlas Ψ on M, computably compatible with Φ and such that for every chart $\psi\colon V \to \mathbb{R}^n$ of Ψ, $\psi[V] = \mathbb{R}^n$.

Proof. (a) For the computable atlas Φ, remember that an element $B_j \in \mathcal{B}_\Phi$ is defined as $B_j = \varphi_i^{-1}[\mu^n_+(w)]$, where $j = \langle i, w\rangle$ and φ_i is a chart of Φ. If $B_j \neq \varnothing$, then we can define a chart $h_j\colon B_j \to \mathbb{R}^n_+$ by $h_j = \varphi_i|_{B_j}$. Let Γ be the set of all such charts, Γ is clearly a computable atlas on M. Now we have to show that Φ and Γ are computably compatible. Notice that if $z = \langle j, w\rangle = \langle\langle i, u\rangle, w\rangle \in \operatorname{dom}\lambda_\Gamma$ is such that $\lambda_\Gamma(z) \neq \varnothing$, then

$$\lambda_\Gamma(z) = h_j^{-1}[\mu^n_+(w)] = (\varphi_i|_{B_j})^{-1}[\mu^n_+(w)] = \varphi_i^{-1}[\mu^n_+(w)] = B_j = \lambda_\Phi(j).$$

and if $v = \langle i, u\rangle \in \operatorname{dom}\lambda_\Phi$ with $\lambda_\Phi(v) \neq \varnothing$, then

$$\lambda_\Phi(v) = \varphi_i^{-1}[\mu^n_+(u)] = (\varphi_i|_{B_v})^{-1}[\mu^n_+(u)] = \lambda_\Gamma(\langle v, u\rangle).$$

There is a Type-2 machine $\mathfrak{T}$ that on input $p \in \operatorname{dom}\delta_\Gamma$, extracts each string $z = \langle\langle i, u\rangle, w\rangle \ll p$, computes the string $l = \langle i, w\rangle \in \operatorname{dom}\lambda_\Phi$ and prints $\iota(l)$ on the output tape. The function $f_\mathfrak{T}$

calculated by $\mathfrak{T}$ translates δ_Γ-names into δ_Φ-names. A Type-2 machine which translates δ_Φ-names into δ_Γ-names is build similarly. This proves that $\delta_\Phi \sim \delta_\Gamma$, thus Φ and Γ are computably equivalent.

(b) If (M, Φ) is a computable manifold with empty boundary, we can replace the space $\mathbf{R}^n_+$ with $\mathbf{R}^n$ to apply definitions and results. Suppose that M has the property of non-empty computable balls, then we can assume that for all $k \in \operatorname{dom}\lambda_\Phi$, $\lambda_\Phi(k) \neq \varnothing$. For each $B_j \in \mathcal{B}_\Phi$ $(j = \langle i, w\rangle)$, define a chart $\psi_j \colon B_j \to \mathbb{R}^n$ as $\psi_j = h_w \circ \varphi_i|_{B_j}$, where h_w is the computable homeomorphism $h_w \colon \mu^n(w) \to \mathbb{R}^n$ of Lemma 3.3.4. The set of charts $\Psi = \{\psi_l \colon B_l \to \mathbb{R}^n\}$ is a topological atlas on M, compatible with Φ. Moreover, Ψ is computable, because the ψ_l's and their inverses are computable with respect to δ_Ψ and δ^n. Also, $\mathbb{R}^n$ is trivially a θ^n-computable open set. We claim that (M, Ψ) is a computable manifold with non-empty computable balls. If $u \in \operatorname{dom}\lambda_\Psi$, then $\lambda_\Psi(u) = \psi_j^{-1}\big[\mu^n(z)\big]$, where $u = \langle j, z\rangle$ and $j = \langle i, w\rangle \in \operatorname{dom}\lambda_\Phi$. Now,

$$\begin{aligned}
\lambda_\Psi(u) \neq \varnothing \quad &\Leftrightarrow \quad \psi_j^{-1}\big[\mu^n(z)\big] \neq \varnothing \\
&\Leftrightarrow \quad (h_w \circ \varphi_i)^{-1}\big[\mu^n(z)\big] \neq \varnothing \\
&\Leftrightarrow \quad j = \langle i, w\rangle \in \operatorname{dom}\lambda_\Phi \wedge \mu^n(z) \subset h_w\big[\varphi_i\big[B_j\big]\big] \\
&\Leftrightarrow \quad j = \langle i, w\rangle \in \operatorname{dom}\lambda_\Phi \wedge \mu^n(z) \subset h_w\big[\mu^n(w)\big] \\
&\Leftrightarrow \quad j \in \operatorname{dom}\lambda_\Phi \wedge \mu^n(z) \subset \mathbb{R}^n \\
&\Leftrightarrow \quad j \in \operatorname{dom}\lambda_\Phi.
\end{aligned}$$

This shows that the set $\{u \in \operatorname{dom}\lambda_\Psi \mid \lambda_\Psi(u) \neq \varnothing\}$ is c.e., so that (M, Ψ) is a computable manifold with non-empty computable balls.

To finish the proof of the theorem, we only need to prove that Φ and Ψ are computably compatible. By Proposition 3.1.12, we can do this by proving that $\mathbf{T}_\Phi(M)$ and $\mathbf{T}_\Psi(M)$ are equivalent computable spaces, which means that we have to show that $\nu_\Phi \leq \theta_\Psi$ and $\nu_\Psi \leq \theta_\Phi$.

Case $\nu_\Phi \leq \theta_\Psi$. An element B of β_Φ is a finite intersection of the form

$$B = \nu_\Phi(\iota(j_1)\cdots\iota(j_k)) = \bigcap_{j \in F} \lambda_\Phi(j) \quad F \subset \operatorname{dom}\lambda_\Phi,$$

where $F = \{j_1, \ldots, j_k\}$. Notice that for any $j \in F$, we have that

$$\begin{aligned}
\lambda_\Phi(j) &= \varphi_i^{-1}\big[\mu^n(w)\big] \qquad (j = \langle i, w\rangle) \\
&= \varphi_i^{-1}\big[h_w^{-1}\big[\mathbb{R}^n\big]\big] \\
&= \varphi_i^{-1}\big[h_w^{-1}\big[\textstyle\bigcup_{z \in \operatorname{dom}\mu^n} \mu^n(z)\big]\big] \\
&= \bigcup_{z \in \operatorname{dom}(\mu^n)} (h_w \circ \varphi_i)^{-1}\big[\mu^n(z)\big] \\
&= \bigcup_{z \in \operatorname{dom}(\mu^n)} \psi_j^{-1}\big[\mu^n(z)\big] \\
&= \bigcup_{z \in \operatorname{dom}(\mu^n)} \lambda_\Psi(\langle j, z\rangle),
\end{aligned}$$

therefore

$$\begin{aligned} B &= \nu_\Phi(\iota(j_1)\cdots\iota(j_k)) \\ &= \bigcap_{j\in F} \lambda_\Phi(j) \\ &= \bigcap_{j\in F}\Big(\bigcup_{z_r\in\operatorname{dom}\left(\mu^n\right)} \lambda_\Psi(\langle j, z_r\rangle)\Big) \\ &= \bigcup_{z_r\in\operatorname{dom}\left(\mu^n\right)} \Big(\bigcap_{j\in F}\lambda_\Psi(\langle j, z_r\rangle)\Big) \\ &= \bigcup_{\substack{z_r\in\operatorname{dom}\left(\mu^n\right)\\ j_l\in F}} \nu_\Psi(\iota(\langle j_{i_1}, z_1\rangle)\cdots\iota(\langle j_{i_k}, z_k\rangle)). \end{aligned}$$

With all this data, we can construct a Type-2 machine that, on input $u = \iota(j_1)\cdots\iota(j_k) \in \operatorname{dom}\nu_\Phi$, computes an element $p_u \in \Sigma^\omega$ such that p_u is a list of all strings $\iota(\langle j_{i_1}, z_1\rangle)\cdots \iota(\langle j_{i_k}, z_k\rangle)$ where $z_l \in \operatorname{dom}(\mu^n)$ and $j_{i_1},\ldots,j_{i_k} \in F$. This reduction works because (M,Φ) and (M,Ψ) have the property of non-empty computable balls (it allows us to avoid having to translate the name of an empty base element of β_Φ into a non-empty element of $\tau_\Phi = \tau_\Gamma$). We conclude that $\nu_\Phi \le \theta_\Psi$.

Case $\nu_\Psi \le \theta_\Phi$. An argument similar to the previous one can be used to prove this case, because for any $l = \langle j, z\rangle \in \operatorname{dom}\lambda_\Psi$,

$$\begin{aligned} \lambda_\Psi(\langle j, z\rangle) &= \psi_j^{-1}[\mu^n(z)] \\ &= (h_w\circ\varphi_i)^{-1}[\mu^n(z)] \qquad (j = \langle i, w\rangle) \\ &= \varphi_i^{-1}[h_w^{-1}[\mu^n(z)]] \\ &= \bigcup_{u\in C_{wz}} \varphi_i^{-1}[\mu^n(u)] \quad (C_{wz} = \{v\in\operatorname{dom}(\mu^n)\mid \mu^n(v)\subset h_w^{-1}[\mu^n(z)]\}) \\ &= \bigcup_{u\in C_{wz}} \lambda_\Phi(\langle i, u\rangle), \end{aligned}$$

where C_{wz} is a computable subset of $\operatorname{dom}(\mu^n)$ (uniformly in w, z, apply part (3) of Lemma 3.3.4). Using this fact, we can construct the Type-2 machine which computes the function that translates λ_Ψ-names into θ_Φ-names. Again, the property of non-empty computable balls is being used to avoid translate the name of an empty element into a non-empty element.

We have proven that $\mathbf{T}_\Psi(M)$ and $\mathbf{T}_\Phi(M)$ are equivalent computable topological spaces. By Proposition 3.1.12, Φ and Ψ are computably compatible atlases on M. The result follows. □

Notice that as every compact computable manifold admits a finite computable atlas, every such manifold has the property of non-empty computable balls, thus (b) of Theorem 3.3.5 is valid for these manifolds. This fact will play an important role in Chapter 4, where we build a computable version of Theorem 1.1.19, which gives an embedding for compact topological manifolds in euclidean spaces.

4 Computable embeddings of manifolds in $\mathbb{R}^q$

In this chapter, we define *computable submanifolds* and prove the computable versions of some basic tools and results that are needed when working with submanifolds. Our main result is a tool that will be most useful when working with computable manifolds. We will construct an *embedding theorem for computable manifolds in euclidean spaces*. Specifically, we will show that any compact computable manifold that is also a Computably Hausdorff manifold, is computably homeomorphic to a computable submanifold of a computable euclidean space $\mathbf{R}^q$, where q depends on the dimension of the manifold. Many versions of embedding theorems exist for all the other types of manifolds, and besides being a powerful tool for proving other stronger results in manifold theory, the embedding theorems show that there is no real difference between abstract manifolds and submanifolds of euclidean spaces.

In Section 4.1, we define *computable submanifolds*. We prove computable versions of some classical results regarding submanifolds (e.g., an open subset of a manifold, is a submanifold; the interior and boundary of a manifold are submanifolds) and then we study computable embeddings of computable manifolds. Section 4.2 contains the main result of this chapter: Any compact computably Hausdorff computable manifold can be embedded in some computable euclidean space. This result is the computable version of the theorem which states that any compact Hausdorff topological manifold can be embedded in some euclidean space.

4.1 Submanifolds and embeddings

One of the most important concepts in the theory of manifolds is that of *submanifold*, that is, a manifold which is a subset of another manifold. In this section, we will develop the corresponding concept of *computable submanifold* and the relations with computable embeddings.

4.1.1 Computable submanifolds

Definition 4.1.1. A computable manifold (M, Φ) is a *computable submanifold* of (N, Ψ) if and only if $M \subset N$ and the inclusion $i\colon M \hookrightarrow N$ is a computable embedding of $\mathbf{T}_\Phi(M)$ into $\mathbf{T}_\Psi(N)$.

A computable submanifold is just a subset of a computable manifold N, which is also a computable manifold in its own right, with computable subspace topology. A simple example of a computable submanifold is the computable n-sphere $\mathbb{S}^n \subset \mathbb{R}^{n+1}$. From the definition, it is clear that the computable topology of a subspace characterizes the computable structure of a computable submanifold. The following result shows an important example of a computable submanifold of a computable manifold.

Proposition 4.1.2. *Let (M,Φ) be a computable manifold and $\varnothing \neq W \subset M$ a θ_Φ-computable open subset of M. Then there exists a computable atlas Ψ on W which converts W into a computable submanifold of M.*

Proof. Let $W \subseteq M$ be any non-empty θ_Φ-computable open subset of (M,Φ). We give W a computable structure induced by that of M. As W is a θ_Φ-computable open set in $\mathbf{T}_\Phi(M)$, there exists a computable infinite string $q_W \in \Sigma^\omega$ such that

$$\theta_\Phi(q_W) = W.$$

For each $j \ll q_W$ such that $\nu_\Phi(j) = V_j$ and $V_j \neq \varnothing$, there exists a chart $(\varphi_i, U_i) \in \Phi$ such that $V_j \subset U$ and $\psi_{\langle i,j\rangle} = \varphi_i|_{V_j} \colon V_j \to \mathbb{R}^n$ is a homeomorphism onto an open subset of $\mathbb{R}^n$, so that the pair $(\psi_{\langle i,j\rangle}, V_j)$ is a chart on W. Let Φ_W be defined by

$$\Phi_W = \{(\psi_{\langle i,j\rangle}, V_j) \mid j \ll q_W \wedge V_j \neq \varnothing \wedge V_j \subset W \cap U_i \wedge \psi_{\langle i,j\rangle} = \varphi_i|_{V_j} \wedge (\varphi_i, U_i) \in \Phi\}.$$

Φ_W is a topological atlas on W, and it can be verified that all charts $\psi_{\langle i,j\rangle}$ with their respective inverses are computable with respect to δ_{Φ_W} and δ^n and the sets $\psi_{\langle i,j\rangle}[V_j]$ are θ^n-computable in $\mathbb{R}^n$. By Definition 3.1.2, Φ_W is a computable atlas on W, thus the pair (W,Φ_W) is a computable n-manifold. Let $\mathbf{T} = \mathbf{T}_\Phi(M)$, we now prove that $W_\mathbf{T}$ and $\mathbf{T}_{\Phi_W}(W)$ are equivalent computable topological spaces by showing that $\delta_W \sim \delta_{\Phi_W}$.

$\delta_{\Phi_W} \leq \delta_W$. For any $k = \langle j,z\rangle \in \operatorname{dom}\lambda_{\Phi_W}$ with $j = \langle i,w\rangle$ and $\lambda_{\Phi_W}(k) \neq \varnothing$, it holds that

$$\lambda_{\Phi_W}(k) = \psi_j^{-1}[\mu^n(z)] = (\varphi_i|_{V_j})^{-1}[\mu^n(z)] = \varphi_i^{-1}[\mu^n(z)] = \lambda_\Phi(\langle i,z\rangle) \subset W.$$

Using the equality $\lambda_{\Phi_W}(k) = \lambda_\Phi(\langle i,z\rangle)$, it is easy to build a Type-2 machine which translates δ_{Φ_W}-names into δ_W-names.

$\delta_W \leq \delta_{\Phi_W}$. This is the part of the proof where the computability of W comes into play, we also need to use a "time-sharing" technique. A Type-2 machine $\mathfrak{M}$ can be build with the following program: On input $p \in \operatorname{dom}\delta_W$, $\mathfrak{M}$ starts enumerating all strings $z \in \Sigma^*$ such that $z = wy$; $w \ll q_W$ and $y \ll p$. This enumeration can be computed by $\mathfrak{M}$, because q_W can be calculated. Notice also that $w \in \operatorname{dom}\nu_\Phi$, $y \in \operatorname{dom}\nu_W = \operatorname{dom}\nu_\Phi$, so that $z \in \operatorname{dom}\nu_\Phi$ and the open set $\nu_\Phi(z)$ is θ_Φ-computable. Let $x = \delta_W(p) = \delta_\Phi(p)$, for each enumerated string z, $\mathfrak{M}$ tries to determine if $x \in \nu_\Phi(z)$, if so, then $\mathfrak{M}$ prints the string $\iota(z')$ on the output tape, where

$$z' = \iota(\langle\langle i_1,e_1\rangle,e_1\rangle)\cdots\iota(\langle\langle i_s,e_s\rangle,e_s\rangle)\iota(\langle\langle j_1,f_1\rangle,f_1\rangle)\cdots\iota(\langle\langle j_r,f_r\rangle,f_r\rangle) \in \operatorname{dom}\nu_{\Phi_W}$$

and $w = \iota(\langle i_1,e_1\rangle)\cdots\iota(\langle i_s,e_s\rangle)$ and $y = \iota(\langle j_1,f_1\rangle)\cdots\iota(\langle j_r,f_r\rangle)$. The machine $\mathfrak{M}$ must execute simultaneously this step for multiple enumerated strings z, and from time to time, $\mathfrak{M}$ must begin executing new tests. This completes the specification of $\mathfrak{M}$.

By part 1 of Lemma 2.1.14, the decision problem "$x \in \nu_\Phi(z)$" is $(\delta_\Phi,\theta_\Phi)$-c.e., thus if $x \in \nu_\Phi(z)$, $\mathfrak{M}$ will finish executing this step for the string z. But if $x \notin \nu_\Phi(z)$, $\mathfrak{M}$ might not be able to finish this part of its program in finite time. This is the reason why $\mathfrak{M}$ must run multiple tests "$x \in \nu_\Phi(z)$" simultaneously, advancing each test a few steps at a time. Because $x \in W$, it cannot happen that all of the test executed by $\mathfrak{M}$ do not succeed. Therefore the output of the machine $\mathfrak{M}$ is non-trivial, it is an infinite string $p' \in \Sigma^\omega$ such that

$$(\forall z' \in \Sigma^*)(z' \ll p' \Leftrightarrow z' \in \operatorname{dom}\nu_{\Phi_W} \wedge \delta_{\Phi_W}(p') \in \nu_{\Phi_W}(z')),$$

and $x = \delta_W(p) = \delta_{\Phi_W}(p')$. Therefore $\delta_W \leq \delta_{\Phi_W}$.

We have shown that $\delta_W \sim \delta_{\Phi_W}$, this says the inclusion $i\colon W \hookrightarrow M$ is a computable embedding of $\mathbf{T}_{\Phi_W}(W)$ onto $\mathbf{T}_\Phi(M)$. By Definition 4.1.1, W is an (open) computable submanifold of M. □

When W is not θ_Φ-computable, we do not know if (W, Φ_W) is a computable submanifold of M, we only can assure that $\nu_W \leq \theta_{\Phi_W}$, that is, everything that is computable in $\mathbf{T}_{\Phi_W}(W)$ is computable in $W_{\mathbf{T}}$. Our next result is an expected and pleasant behavior from the interior and boundary of a computable manifold.

Proposition 4.1.3. *Let (M, Φ) be a computable n-manifold with boundary. Then ∂M and* $\operatorname{Int} M$ *are computable submanifolds of M.*

Proof. We first prove that $\operatorname{Int} M$ is a computable submanifold of M. From Lemma 2.1.9, we know that $\operatorname{Int} \mathbb{R}^n_+$ is θ^n-computable and $(\theta^n)^-$-computable in $\mathbf{R}^n$ (where θ^n and $(\theta^n)^-$ are the positive and negative information representations of open subsets of $\mathbb{R}^n$, see Definitions 2.1.6 and 2.1.7) and this can be used to build a Type-2 machine $\mathfrak{T}$ that does the following: $\mathfrak{T}$ enumerates all the elements of $\operatorname{dom} \lambda_\Phi$; for each string $j = \langle i, w \rangle \in \operatorname{dom} \lambda_\Phi$, $\mathfrak{T}$ determines if $\mu^n_+(w) \subset \operatorname{Int} \mathbb{R}^n_+$ (μ^n_+ is the notation for base elements of $\mathbb{R}^n_+$, see Example 2.1.19). If the test succeed, $\mathfrak{T}$ outputs $\iota(j)$. Clearly, $\mathfrak{T}$ computes a string $q \in \Sigma^\omega$ such that $\theta_\Phi(q) = \operatorname{Int} M$, so that the interior of M is θ_Φ-computable. By Proposition 4.1.2, $\operatorname{Int} M$ is a computable submanifold of M.

Now to show that ∂M is a computable submanifold. Let $\partial\mathbf{R}^n_+ = (\partial\mathbb{R}^n_+, \partial\tau^n, \partial\beta^n, \partial\mu^n)$ be the computable space of Example 2.1.19, $p\colon \mathbb{R}^n \to \mathbb{R}^{n-1}$ the computable projection $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_{n-1})$ and $p_0 = p|_{\partial\mathbb{R}^n_+}$ the canonical computable homeomorphism from $\partial\mathbb{R}^n_+$ to $\mathbb{R}^{n-1}$. Using the homeomorphism p_0, we can replace the standard base β^{n-1} of $\mathbf{R}^{n-1}$ with the base $\beta_{p_0} = \{p_0[B] \mid B \in \partial\beta^n\}$ of the computable space $\mathbf{R}_{p_0} = (\mathbb{R}^{n-1}, \tau^{n-1}, \beta_{p_0}, \nu_{p_0})$ induced by p_0 (see Corollary 2.2.16). Let Φ' be the atlas

$$\Phi' = \{(g_i, V_i) \mid V_i = U_i \cap \partial M \neq \varnothing \wedge g_i = p_0 \circ \varphi_i|_{V_i} \wedge (\varphi_i, U_i) \in \Phi\}.$$

The pair $(\partial M, \Phi')$ becomes a submanifold (without boundary) of M, of dimension $n-1$. We only need to show that it is a computable submanifold. It is easy to check that Φ' satisfies definition 3.1.2, because Φ is computable and the set $\partial\mathbb{R}^n_+$ is ψ^n-computable in $\mathbf{R}^n$. Let $\mathbf{T} = \mathbf{T}_\Phi(M)$. We now prove that the spaces $\partial M_{\mathbf{T}}$ and $\mathbf{T}_{\Phi'}(\partial M)$ are equivalent. If $z = \langle i, w \rangle \in \operatorname{dom} \lambda_{\Phi'}$ is such that $\lambda_{\Phi'}(z) \neq \varnothing$, then notice that

$$\begin{aligned}
\lambda_{\Phi'}(z) &= g_i^{-1}[p_0[\mu^n_+(w)]] \\
&= (p_0 \circ \varphi_i)^{-1}[p_0[\mu^n_+(w)]] \\
&= \varphi_i^{-1} \circ p_0^{-1}[p_0[\mu^n_+(w)]] \\
&= \varphi_i^{-1}[\mu^n_+(w)] \\
&= \lambda_\Phi(z).
\end{aligned}$$

Using this property, we deduce that, for the representations $\delta_{\partial M}, \delta_{\Phi'}$ and $p \in \Sigma^\omega, x \in \partial M$,

$$\begin{aligned}
\delta_{\Phi'}(p) = x &\iff (\forall w \in \Sigma^*)(w \ll p \Rightarrow w \in \operatorname{dom} \lambda_{\Phi'} \wedge x \in \lambda_{\Phi'}(w)) \\
&\iff (\forall w \in \Sigma^*)(w \ll p \Rightarrow w \in \operatorname{dom} \lambda_{\Phi} \wedge x \in \lambda_{\Phi}(w)) \\
&\iff \delta_\Phi(p) = x,
\end{aligned}$$

and this implies that $\delta_{\partial M} \sim \delta_{\Phi'}$, so that applying Theorem 2.2.13, we have that $\partial M_{\mathbf{T}}$ and $\mathbf{T}_{\Phi'}(\partial M)$ are equivalent, that is, the inclusion of ∂M is a computable embedding of $\mathbf{T}_{\Phi'}(\partial M)$ onto $\partial M_{\mathbf{T}}$, so that ∂M is a computable submanifold of M. □

Example 4.1.4. If we give $\mathbb{S}^1$ and D^2 the computable structures defined in Examples 3.1.9 and 3.1.17 respectively, then using Proposition 3.1.19, the *solid torus* $N = \mathbb{S}^1 \times D^2$ is a computable manifold of dimension 3, it has a boundary ∂N, which by Lemma 1.1.18, is equal to $(\partial\mathbb{S}^1 \times D^2) \cup (\mathbb{S}^1 \times \partial D^2) = \mathbb{S}^1 \times \mathbb{S}^1$, thus the boundary of N is the torus. By Proposition 4.1.3, ∂N has a computable structure of dimension 2. It can be seen that the computable structure of ∂N induced by the computable structure of N, is the same structure imposed to the torus in Example 3.1.20.

4.1.2 Embeddings of computable manifolds

In Section 2.2, we introduced computable embeddings of computable topological spaces, the next lemma shows their relationship with computable manifolds. Informally, it says that every computable manifold that is computably imbedded in another computable manifold, can be thought as a computable submanifold of the latter.

Lemma 4.1.5. *Let $(M^n, \Phi), (N^m, \Psi)$ be computable manifolds and $h\colon M \hookrightarrow N$ be a computable embedding. Then the subset $M' = h[M]$ has the structure of a computable submanifold of N, such that $M \cong_{ct} M'$.*

Proof. Let $\mathbf{T} = \mathbf{T}_\Psi(N)$. By Corollary 3.2.5, the computable homeomorphism $h\colon M \to M'$ induces a computable structure Φ_h on M' such that $\mathbf{T}_{\Phi_h}(M')$ is equivalent to $M'_{\mathbf{T}}$, therefore the inclusion of M' into N is a computable embedding of $\mathbf{T}_{\Phi_h}(M')$ into $\mathbf{T}$, so that (M', Φ_h) is a computable submanifold of (N, Ψ). As $\mathbf{T}_\Phi(M) \cong_{ct} \mathbf{T}_{\Phi_h}(M')$, we are done. □

Example 4.1.6. In example 3.1.20, if we take $n = 2$, the 2-dimensional Torus $T^2 = \mathbb{S}^1 \times \mathbb{S}^1$ is a computable submanifold of $\mathbb{R}^4$. But it is known from standard topology that the map $h\colon T^2 \hookrightarrow \mathbb{R}^3$ given by

$$h(x_u, y_u, x_v, y_v) = ((2 + y_v)y_u, (2 + y_v)x_u, x_v),$$

is a homeomorphism of T^2 onto the set $T' = h[T^2]$ (that is, it is an embedding). Clearly, h is a computable embedding of $\mathbf{T}_\Phi(T^2)$ into $\mathbf{R}^3$. By Lemma 4.1.5, h induces a computable structure on T', such that $T'_{\mathbf{R}^3}$ is equivalent to $\mathbf{T}_{\Phi_h}(T')$, that is, (T', Φ_h) becomes a computable submanifold of $\mathbf{R}^3$.

Lemma 4.1.7. *Every computable submanifold of a computably Hausdorff computable manifold is a computably Hausdorff manifold.*

Proof. This is an immediate consequence of Theorem 2.3.3 and Definition 4.1.1. □

4.2 Embedding computable manifolds in euclidean spaces

There are many facts [KS77] in the theory of manifolds (topological, differentiable and/or PL) which can be shown to be true using the well known result that every Hausdorff n-manifold M (of any kind) embeds in some high dimensional euclidean space $\mathbb{R}^q$, where q depends on n. Many versions of this embedding theorem exist, being the big difference between them the dimension of the space $\mathbb{R}^q$. It was proven by Whitney [Whi36] that if M is smooth, then it embeds in $\mathbb{R}^{2n}$ and this is the best possible result. The same is true for the piecewise linear case using similar constructions to those used in the smooth case. If M has no additional structure, it can be embedded in $\mathbb{R}^{2n+1}$

and again, this is the lowest possible dimension for the euclidean space. This last result can be proven by means of dimension theory [HW48, Eng95, Mun00].

Our aim now is to prove a computable version of an embedding theorem for compact computable manifolds which are computably Hausdorff. We will show that every such manifold can be imbedded in computable euclidean space $\mathbf{R}^q$ (where q must be large enough, we will not try to optimize q) with a computable embedding. The question remains open if a compact computably Hausdorff computable manifold can be computably embedded in a lower dimensional euclidean space. Notice that by Lemma 4.1.7, any computable manifold embedded in computable euclidean space must have the property of being computably Hausdorff.

4.2.1 A simple embedding theorem

From now on, all computable manifold are assumed to be computably Hausdorff. We now show that every compact computably Hausdorff computable manifold has an embedding in some euclidean space of sufficiently high dimension. We develop a computable version of the proof of Theorem 1.1.19 which can be found in [Gau74].

Let (M, Φ) be a compact computable manifold of dimension n and suppose that Φ is the atlas given in part (a) of Theorem 3.3.5. Let $(\varphi, U) \in \Phi$, where $\varphi[U] = \mu^n_+(z) \cap \mathbb{R}^n_+$ for some $z \in \operatorname{dom}(\mu^n_+)$. Let the map $f_z\colon \mu^n_+(z) \to \mathbb{R}^n$ be given as follows: $f_z = h_z$ if $\mu^n_+(z) \subset \operatorname{Int}\mathbb{R}^n_+$ and $f_z = g_z$ if $\mu^n_+(z) \cap \partial\mathbb{R}^n_+ \neq \varnothing$, where h_z, g_z are the computable homeomorphisms of Lemma 3.3.4. Let $s\colon \mathbb{S}^n - \{P\} \to \mathbb{R}^n$ ($P = (0, \dots, 0, 1)$) be the computable stereographic projection given in Example 2.2.10. Define a function $g\colon M \to \mathbb{S}^n$ by

$$g(x) = \begin{cases} s^{-1} \circ f_z \circ \varphi(x) & \text{if } x \in U, \\ P & \text{if } x \in M - U. \end{cases}$$

It is not hard to show that g is continuous. We now prove that

Lemma 4.2.1. *The function g is computable.*

Proof. We prove that g is a computable map from $\mathbf{T}_\Phi(M)$ to $\mathbf{S}^n$ by showing that the map $B \mapsto g^{-1}[B]$ is $(\nu_{\mathbb{S}^n}, \theta_\Phi)$-computable (See Definition 2.2.4). There is a Type-2 machine $\mathfrak{M}$ that on input $V = B \cap \mathbb{S}^n$ ($B = B(r, \epsilon) \subset \mathbb{R}^{n+1}$), does the following:

1. If V contains the point P, then execute the following steps:
    1.1 Calculate the compact set $K = \mathbb{S}^n - V$;
    1.2 compute the compact set $K' = g^{-1}[K]$;
    1.3 output the set $V' = M - K'$.
2. If V does not contains P, output the open set $g^{-1}[V]$.

First, we show that each step can be executed by $\mathfrak{M}$. The test $P \in V$ in steps 1 and 2 can be done by $\mathfrak{M}$ in finite time, because $P \in \mathbb{Q}^n$ and $V = B \cap \mathbb{S}^n$ is specified by rational numbers. To construct the set K in step 1.1, $\mathfrak{M}$ needs to compute an element $p \in \Sigma^\omega$ such that $K \in \kappa_{\mathbb{S}^n}(p)$ and this can be done by $\mathfrak{M}$ because the map $V \mapsto \mathbb{S}^n - V$ is $(\nu_{\mathbb{S}^n}, \kappa_{\mathbb{S}^n})$-computable. Step 1.2 can be accomplished because in the subset $U \subset \operatorname{dom} g = M$, $g^{-1} = \varphi^{-1} \circ f_z^{-1} \circ s$ exists and it is a computable function, thus the function $K \mapsto g^{-1}[K]$ is $(\kappa_{\mathbb{S}^n}, \kappa_\Phi)$-computable. To execute step 1.3, $\mathfrak{M}$ can use the computable function $f\colon \subseteq \Sigma^\omega \to \Sigma^\omega$ of the reduction $\kappa_\Phi \le \psi^-_\Phi$. This reduction exists because $\mathbf{T}_\Phi(M)$ is computably Hausdorff (part 4 of Theorem 2.3.3). The construction of the set $g^{-1}[V]$ in step 2 can be executed by $\mathfrak{M}$ because as $P \notin V$, $V \subset \mathbb{S}^n - \{P\}$, thus in V, g is a computable function. This proves that each step of $\mathfrak{M}$ can be done in finite time, so that $\mathfrak{M}$ is a valid Type-2 machine.

We now prove the correctness of this pseudocode. Assume that $P \in V$, then $\mathfrak{M}$ goes on to execute step 1.1 and compute the set $K = \mathbb{S}^n - V$. Notice that as $P \in V$, $K \subset g[U]$, so that in K, $g = s^{-1} \circ f_z \circ \varphi$ and $\mathfrak{M}$ can use s, f_z and φ^{-1} to compute the compact set $K' = g^{-1}[K] \subset U$ in step 1.2. Finally, in step 1.3, $\mathfrak{M}$ computes the open set $V' = M - K'$, and this set is such that $V' = g^{-1}[V]$. Suppose that $P \notin V$. Then $V \subset \mathbb{S}^n - \{P\}$ and $\mathfrak{M}$ uses g to compute the open set $g^{-1}[V] \subset U$.

This proves that the machine $\mathfrak{M}$ computes a function which realizes the map $V \mapsto g^{-1}[V]$ with respect to $\nu_{\mathbb{S}^n}$ and θ_Φ. Therefore, the map $V \mapsto g^{-1}[V]$ is $(\nu_{\mathbb{S}^n}, \theta_\Phi)$-computable, thus by Definition 2.2.4, g is a computable function from M to $\mathbb{S}^n$. □

We are ready to prove the main result of this chapter.

Theorem 4.2.2. *For any compact computable manifold M^n, there exists a computable embedding of M into $\mathbf{R}^q$ for q sufficiently large.*

Proof. Let M be a compact computable manifold of dimension n and assume that $\mathbb{S}^n$ has the computable subspace topology induced by $\mathbf{R}^{n+1}$. By compactness of M and part (a) of Theorem 3.3.5, we can find a finite atlas $\{(\varphi_1, U_1), \ldots, (\varphi_l, U_l)\}$ such that $\varphi_i[U_i] = \mu^n_+(z_i) \cap \mathbb{R}^n_+$ for all $i = 1, \ldots, l$. Using Lemma 4.2.1, we construct computable functions $g_i\colon M \to \mathbb{S}^n$. Now let

$$X = \underbrace{\mathbb{S}^n \times \cdots \times \mathbb{S}^n}_{l \text{ times}} \quad \text{and} \quad \overline{\mathbf{X}} = (X, \overline{\tau}, \overline{\beta}, \overline{\nu})$$

and define $G\colon M \to X$ of $\mathbf{T}_\Phi(M)$ into $\overline{\mathbf{X}}$ by $G = (g_1, \ldots, g_l)$. We now prove that G is injective. Take $x, y \in M$ with $x \neq y$. If there is a chart U_i such that $x, y \in U_i$ then $g_i(x) \neq g_i(y)$ because g_i is inyective in U_i. If x and y are not in the same chart, then $x \in U_i$ for some i and $y \notin U_i$, hence $g_i(x) \neq P$ and $g_i(y) = P$. Since M is compact and X is Hausdorff, then G is closed, therefore G is a topological embedding. Because each g_i is computable, G is computable (use part 2. of Lemma 2.1.20). To see that the inverse function G^{-1} is computable, let $\mathfrak{M}'$ be a Type-2 machine that on input $x \in \operatorname{range} h$, executes the following steps:

1. Compute each component $x_i \in \mathbb{S}^n$ $(i = 1, \ldots, l)$ of x.
2. Find j such that $x_j \in \mathbb{S}^n - P$.
3. Output $y = g_j^{-1}(x_j)$.

Step 1 can be computed because by Lemma 2.1.20, the map $x \mapsto x_i$ is $(\overline{\delta}, \delta_{\mathbb{S}^n})$-computable for each i; step 2 is terminated in finite time because each $x_i \in \mathbb{S}^n \subset \mathbb{R}^{n+1}$ is $\delta_{\mathbb{S}^n}$-computable, the set $\mathbb{S}^n - P$ is $\theta_{\mathbb{S}^n}$-computable open in $\mathbb{S}^n$ and by part 1 of Lemma 2.1.14, the decision problem "$x_i \in \mathbb{S}^n - P$" is $(\delta_{\mathbb{S}^n}, \theta_{\mathbb{S}^n})$-c.e. ($x_j \neq P$ for at least one index j); step 3 is easily calculated because g_j^{-1} exists and it is $(\delta_{\mathbb{S}^n}, \delta_\Phi)$-computable in $\mathbb{S}^n - P$. Hence, the function computed by $\mathfrak{M}'$ realizes g^{-1}, that is, it is a computable function.

Thus G is a computable embedding of $\mathbf{T}_\Phi(M)$ into $\overline{\mathbf{X}}$, which is a computable subspace of $\overline{\mathbf{X}}' = (X', \overline{\tau}', \overline{\beta}', \overline{\nu}')$ ($X' = \prod_i^l \mathbb{R}^{n+1}$, each factor is to be understood as the computable space $\mathbf{R}^{n+1}$), which is equivalent to the computable euclidean space $\mathbf{R}^{l(n+1)}$, so that by combining G with the computable inclusion of $\overline{\mathbf{X}}$ into $\overline{\mathbf{X}}'$ and the equivalence of $\overline{\mathbf{X}}'$ with $\mathbf{R}^{l(n+1)}$, we obtain the desired computable embedding of M into $\mathbb{R}^{l(n+1)}$. □

With the computable embedding constructed in Theorem 4.2.2, we can deduce that abstract compact computably Hausdorff computable manifolds and compact computable submanifolds of euclidean spaces are essentially the same.

Conclusions for the first part of this thesis can be found in Chapter 10.

Part II:
Distributed systems

With every choice we make, we literally create a world. History branches in two, creating a world where we made the choice; and a second one, where we did not. That is the secret of the universe you know, billions of people taking billions of choices, creating infinite worlds.

Owlman when talking to Batman in Justice League: Crisis in two earths.

5 The iterated shared memory model of distributed computing

We begin the second part of this thesis. Here we give an introduction to the field of *distributed computability*, a natural evolution of standard computability theory. Specifically, we will be talking about solvability of tasks in some kind of distributed system and how we can derive results for these systems using topology and combinatorics. To do this, a well known concept of distributed computing is necessary: *Iterated shared memory models*. The word "iterated" comes from the fact that in an iterated model, computations proceed in a *round-based* pattern and in each iteration of the system, some communication objects are used and each object can be used at most once. In this chapter, we explain the main ideas behind the concept of iterated models and show the beautiful connection that distributed systems share with topology. Also, we use the iterated model and topology to explain some popular results which are based on iterated models, topology and/or combinatorics.

In Section 5.1, we introduce the basic definitions regarding models of distributed systems. In Section 5.2 we define two widely used models of distributed computing: The *standard model* and the *iterated shared memory model*, being the latter the model in which we will focus all our attention. Section 5.3 describes how topology can be used to give a geometric description of the executions of a protocol in the iterated model, using simplicial complexes. Section 5.4 introduces the definition of the two distributed tasks studied in this thesis: The *set agreement* task and the *consensus* task. At the end of the section, we give a proof that the consensus task cannot be implement in the standard model of distributed computing, using combinatorics and the iterated model.

5.1 Basic definitions

Our formal model is standard, we follow the usual definitions given in [CR12], which are based on concepts and ideas taken from [AR02].

5.1.1 Distributed systems

The first thing that we need to define formally is the concept of a system, which is, roughly, a set of processes (computers, network nodes, sensors) and some communication media, in our case, the communication media is a memory which is visible to all the processes.

Definition 5.1.1. A *process* is a deterministic state machine[1], which has a (possible infinite) set of *local states*, including a subset called the *initial states* and a subset called the *output states*.

[1]See [HU90] or [HS99, p. 863, Definition 2.2] for the formal definitions of state machines and I/O automata.

Definition 5.1.2. A *system* is a pair (Π, MEM), which consists of a set $\Pi = \{p_1, \ldots, p_n\}$ of $n \geqslant 1$ processes and a set $MEM = \{R_1, \ldots, R_m\}$ of $m \geqslant 1$ *shared registers*. Each register has a type, which specifies the following data:

1. The values that can be taken on by the register.
2. The operations that can be performed on the register.
3. The value to be returned by each operation (if any).
4. The new value of the register resulting from each operation.

The domain of values of each register can include a special "undefined" value, which we denote by $\perp$. Given a process $p_i \in \Pi$, the subindex i is the *id* of process p_i.

Intuitively, a register is an abstract object which can hold any element of a fixed set (the type of the register). It is the formal way to represent a variable of some programming language (i.e., Java, C, Fortran). The operations defined on the register allow us to change the element it is storing with another member of the set of possible values (change the state of the register) or know the current value of the element in the register. For example, an integer valued read/write register R can take on all integer values and has operations $R.\mathsf{read}()$ and $R.\mathsf{write}(v)$. The read operation returns an integer value u, which is stored in R and leaves R unchanged. The write operation takes an integer input parameter v, returns no value, and changes R's value to v. The word shared comes from the fact that all processes can read and modify the contents of a shared register.

It is also customary to make no assumptions about the size of the registers of the shared memory, and therefore we may assume that each process p_i can write its entire local state in a single register.

Definition 5.1.3. Let (Π, MEM) be a system. A *global state* is a vector

$$S = \langle s_1, \ldots, s_n; MEM \rangle,$$

where s_i is the local state of process $p_i \in \Pi$. An *initial state* is a state in which every local state is an initial local state and all register in the shared memory are set to $\perp$. A *decision state* is a state in which all local states are output states.

5.1.2 Events and round schedules

The occurrences that can take place in a system are modeled as *events*, these are basic actions which are performed by the processes. The events affect the state of the shared memory of the system. We now describe the main events to be consider in our work.

Let (Π, MEM) be a system. An *event* in the shared memory is performed by a single process $p_i \in \Pi$, which applies only one of the following actions: a write (W) or read (R) operation on a register $R_j \in MEM$. Any of these operations may be preceded/followed by some local computation, formally a change of the process to its next local state. We will need to consider events performed *concurrently* by the processes. If E is any event and $p_{i_1}, \ldots, p_{i_k} \in \Pi$ are processes, then we denote the fact that $p_{i_1}, \ldots, p_{i_k}$ execute concurrently the event E by $\mathsf{E}(X)$, where $X = \{i_1, \ldots, i_k\}$.

We fix once and for all some notation. Let $\overline{n} = \{1, \ldots, n\}$, when convenient, we will denote $\mathsf{E}(X)$ by $\mathsf{E}(i_1, \ldots, i_k)$ and if $i \in \overline{n}$ is a process id, then $\mathsf{E}(\overline{n} - \{i\})$ is written simply as $\mathsf{E}(\overline{n} - i)$.

Definition 5.1.4. Given a system (Π, MEM), a *round schedule* π is a finite sequence of events of the form

$$\pi\colon \mathsf{E}_1(X_1), \ldots, \mathsf{E}_\mathsf{r}(X_r),$$

that encodes the way in which the processes with ids in the set $\bigcup_j^r X_j$ perform the events $\mathsf{E}_1, \ldots, \mathsf{E}_r$.

For example, the round schedule given by

$$\mathsf{W}(1,3), \mathsf{R}(1,3), \mathsf{W}(2), \mathsf{R}(2),$$

means that processes p_1, p_3 perform the write and read events concurrently; after that, p_2 executes in solo its shared memory events. Similarly, the round schedule $\mathsf{W}(1,2,3), \mathsf{R}(1,2,3)$ says that p_1, p_2, p_3 execute concurrently the write and read events.

5.1.3 Protocols and executions

Definition 5.1.5. Let (Π, MEM) be a system. The state machine of each process $p_i \in \Pi$ is called a *local protocol* $\mathcal{A}_i$. A *protocol* is a collection $\mathcal{A}$ of local protocols $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

We assume that all local protocols are identical; i.e., processes have the same state machine. For the sake of simplicity, we will give protocols specifications using pseudocode and we establish the following conventions: A lowercase variable denotes a local variable, with a subindex that indicates to which process it belongs; the shared memory (which is visible to all processes) is denoted with uppercase letters. Intuitively, the local state s_i of process p_i is composed of the contents of all the local variables of p_i.

Remarks. From now on, we focus all our attention on protocols, and when we say something about a protocol $\mathcal{A}$, we will omit an explicit reference to the system (Π, MEM) behind $\mathcal{A}$, if there is no confusion.

Definition 5.1.6. Let $\mathcal{A}$ be a protocol. An *execution* of $\mathcal{A}$ is a finite or infinite alternating sequence of states and round schedules

$$S_0, \pi_1, \ldots, S_k, \pi_{k+1}, \ldots,$$

where S_0 is an initial state and for each $k \geqslant 1$, S_k is the resulting state of applying the sequence of events performed by the processes in the way described by the round schedule π_k. An *r-round partial execution* of $\mathcal{A}$ is a finite execution of $\mathcal{A}$ of the form $S_0, \pi_1, \ldots, S_{r-1}, \pi_r, S_r$, that is, an execution of $\mathcal{A}$ until the end of round r.

If P is a state, P is said to be *reachable in* $\mathcal{A}$ if there exists an r-round partial execution of $\mathcal{A}$ ($r \geqslant 0$) that ends in the state P and when there is no confusion about which protocol we refer to, we just say that S is reachable. Also, we identify two special components of each process' states: an input and an output. It is assumed that initial states differ only in the value of the input component; moreover, the input component never changes. The protocol cannot overwrite the output, it is initially $\perp$; once a non-$\perp$ value is written to the output component of the state, it never changes; when this occurs, we say that the process *decides*. The output states are those with non-$\perp$ output value.

Definition 5.1.7. Let $\mathcal{A}$ be a protocol and S, R global states, we say that R is a *successor* of S in $\mathcal{A}$, if there exists an execution α of $\mathcal{A}$ such that

$$\alpha = S_0, \pi_1, \ldots, S_r = S, \pi_{r+1}, \ldots, \pi_{r+k}, S_{r+k} = R, \ldots,$$

i.e., starting from S, we can run the protocol $\mathcal{A}$ k rounds (for some $k \geqslant 0$) such that the system enters state R.

If π is any round schedule and S is a state, the successor of S in $\mathcal{A}$ obtained by running the protocol (starting in the state S) one iteration with the round schedule π is denoted by $S \cdot \pi$.

Definition 5.1.8. Two states S, P are said to be *adjacent* if there exists a non-empty subset $X \subseteq \overline{n}$ such that all processes with ids in X have the same local state in both S and P. That is, for each $i \in X$, p_i cannot *distinguish* between S and P. We denote this by $S \overset{X}{\sim} P$. States S and P are *connected*, if we can find a sequence of states (called a *path*)

$$\mathfrak{p}\colon S = P_1 \sim \cdots \sim P_r = P,$$

such that for all j with $1 \leqslant j \leqslant r-1$, P_j and P_{j+1} are adjacent.

Connectivity of global states are a key concept for many beautiful results in distributed systems, namely, impossibility proof. The indistinguishability of states between processes is the building block to construct topological structures based on the executions of a given protocol and is fundamental in many papers [LAA87, HS99, SZ00, BG93a, CR12].

5.1.4 Decision tasks

In distributed computing, a *decision task* is a problem that must be solved in a distributed system. Each process starts with a private input value, communicates with the others, and halts with a private output value. The nature of many tasks that arise often in asynchronous systems has suggested that the best way to model decision tasks formally is by using relations instead of functions (In contrast with the case of deterministic single process Turing computability [Sip96, Koz97, Coo04]). A given input value of a task in distributed systems may have more than one acceptable output value from the processes. We now introduce the formal concept of a decision task and define how and when protocols solve decision task.

Definition 5.1.9. A *decision task* Δ is a relation that has a domain $\mathcal{I}$ of input values and a domain $\mathcal{O}$ of output values; Δ specifies for each assignment of the inputs to processes on which outputs processes can decide. A *bounded* decision task is a task whose number of input values is finite.

We also refer to decision task simply as tasks. Examples of tasks includes *consensus* [FLP85], *renaming* [BG93b, AGL$^+$08] and the *set agreement* task [Cha93] (to be defined later in Section 5.4). One of the main goals in distributed computing, is to design efficient protocols that can solve decision tasks like the examples given above and many others.

Definition 5.1.10. The protocol $\mathcal{A}$ *solves* a decision task Δ if any finite execution α of $\mathcal{A}$ can be extended to an execution α' in which all processes decide on values which are allowable (accordingly to Δ) for the inputs in α.

Because the outputs cannot be overwritten, if a process has decided on a value in α, it must have the same output in α'. This means that outputs already written by the processes can be completed to outputs for all processes that are permissible for the inputs in α.

Definition 5.1.11. A protocol $\mathcal{A}$ is *wait-free* if in any execution of $\mathcal{A}$, a process either it has a finite number of events or it decides. This implies that if a process has an infinite number of events, it must decide after a finite number of events.

Roughly speaking, $\mathcal{A}$ is wait-free if any process that continues to run will halt with an output value in a fixed number of steps, regardless of delays or failures by other processes. However, in our formal model, we do not require the processes to halt; they solve the decision task and decide by writing to the output component; processes can continue to participate. We typically consider the behavior of a process until it decides, and therefore, the above distinction does not matter. The study of wait-free shared memory protocols has been fundamental in distributed computing, some of the most powerful results have been constructed on top of wait-free protocols [BG93a, HS99, SZ00, Raj10]. Also, other variants of distributed systems can be reduced to the wait-free case [BG93a, Gaf09, BGLR01].

5.2 Two popular models of distributed computing

Intuitively, a model of distributed computing describes a set of protocols that share some common properties and/or restrictions in the way the processes can access the shared objects and these restrictions affect the way in which the protocols can be specified. We will consider two shared memory distributed computing models, the first one is the usual read/write model and the second is an iterated read/write model. This model has nice properties, because protocols in this model can be executed by the processes in a round-based fashion and the shared objects are used one after the other and in the same order. We will have something very important to say about the relationships between these two popular models. In later chapters, we will extend these models with more powerful objects that shared memory.

5.2.1 The standard model

In the *standard shared memory model of distributed computing* [HW90, Her91], there are n processes that communicate by reading and writing all the information they have (the entire state of each process) in a shared memory. This shared memory is composed of *single-writer, multi-reader* registers, that is, any number of processes can read and write concurrently a single register of the memory. For protocols defined in the standard model, there are no restrictions on the way the processes can access the registers, thus it is the more general model, although some simple and useful restrictions can be assumed without loss [AAD$^+$93]. There are many examples of protocols in the standard model in the literature [HW90, Her91, AAD$^+$93, AW04, AGL09, GR10a].

A drawback of the standard model, is that while there is a lot of freedom to specify a protocol to solve a distributed task, it is very hard to obtain impossibility results. This is because in general protocols in the standard model are very wild and one cannot make assumptions about the way the executions of a given protocol unfold in time, there is no way to assure that an execution will follow some order or pattern in the way the processes execute the operations in the shared objects and because of this, pursuing what happens with a protocol in a formal way is error prone. Subtle mistakes in the impossibility proofs often can go unnoticed by researchers.

5.2.2 The iterated model

Because of the difficulties to build impossibility proofs in the standard model, it is necessary to impose some restrictions in the way the processes execute the operations on the shared objects, but of course, we need to do this without losing the computational power of the standard model. This is accomplish with the second model that we present in this section: The *iterated model of distributed computing*. In this model of computation, the processes communicate asynchronously through a sequence of shared objects, called *snapshots*, these objects can be accessed each one at most once by each process and also all the snapshots are used in the same order by all the processes.

Definition 5.2.1. A one-shot snapshot object S is a shared memory array $S[1,\ldots,n]$ with one entry per process. That array is initialized to $[\perp,\ldots,\perp]$, where $\perp$ is a default value that cannot be written by a process. The snapshot object S provides two atomic operations that can be used by a process at most once:

- $S.\mathsf{update}(v)$: when called by process p_j, it writes the value v to the register $S[j]$.
- $S.\mathsf{scan}()$: returns a copy of the whole shared memory array S.

It is proven in [AAD+93, BG93b] that snapshot objects can be wait-free implemented using only read/write shared registers. Snapshot objects are the principal shared objects that we use to define our next distributed model.

Definition 5.2.2. Let (Π, SM) be a system, where the shared memory SM is structured as an infinite sequence of one-shot snapshot objects $SM[i]$ $(i \geqslant 0)$. We say that the protocol $\mathcal{A}$ is a *protocol in the iterated read/write shared memory model* if each local protocol $\mathcal{A}_i$ of $\mathcal{A}$ can be written in the form given in Figure 5.1.

```
(1)  init $r \leftarrow 0; sm \leftarrow input; dec \leftarrow \perp;$

(2)  loop forever
(3)       $r \leftarrow r + 1;$
(4)       $SM[r].\mathsf{update}(sm);$
(5)       $sm \leftarrow SM[r].\mathsf{scan}();$

(6)       if $(dec = \perp)$ then
(7)            $dec \leftarrow \delta(sm);$
(8)       end if
(9)  end loop
```

Figure 5.1: General form of a protocol in the iterated model

We now give an intuitive explanation of the pseudocode[2] of Figure 5.1. Initially, r is zero and sm is assigned the contents of the readonly variable $input$, which contains the input value for process p_i; all other variables are initialized to $\perp$. In each iteration, p_i increments by one the loop counter r, accesses the current shared memory array $SM[r]$, writing all the information it has

[2]Remember what we said in the paragraph following Definition 5.1.5 about pseudocodes: All the lowercase variables $r, sm, input$ and dec are local to process p_i and only when we analyze a protocol, we add a subindex i to a variable to specify it is local to p_i.

stored in *sm* and after this, p_i takes a snapshot of the shared array (which contains data from p_i and possibly from other running processes) and finally, p_i checks if *dec* is equal to $\perp$, if so, then with all its information, it enters into a deterministic function δ to determine if it must return a valid output value or $\perp$, which is stored in *dec*. Once a non-$\perp$ value is assigned to *dec*, it is never overwritten.

As in the case of the standard model, there are plenty of examples of protocols in the iterated model [BG93b, BG97, Gaf09, GR10a]. In later chapters, we will present examples of iterated protocols in a new model of distributed computing, which is an extension of the basic iterated model given in this section.

The equivalence of the iterated model with the standard model

We finish this section with some remarks about protocols in the iterated model. It would seem that the iterated model that we have defined is too restrictive: a process cannot go back and read again the same shared array. Moreover, all processes must access all the shared arrays in the same order. Because of this, one may think that the iterated model does not have full generality when compared with the more standard, non-iterated wait-free shared memory model, that does not have the restrictions imposed to protocols in the iterated model. The crucial question is: Does there exists a task that is solvable in the wait-free standard model and it is not solvable in the iterated model? The answer is no. Every task that is solvable in the wait-free standard model is also solvable in the iterated model. This is proved using an algorithm that simulates the standard model in the iterated model, first in [BG97], and more recently in [GR10a].

Theorem 5.2.3 (Borowsky and Gafni)**.** *If a bounded decision task can be wait-free solved in the standard model then it can be wait-free solved in the iterated model.*

Of course, the other direction of the theorem is trivial. Therefore, the standard model and the iterated model of distributed computing are equivalent, as long as task solvability is concerned, so that there is no loss of generality (for computability purposes) in considering only protocols in the iterated model.

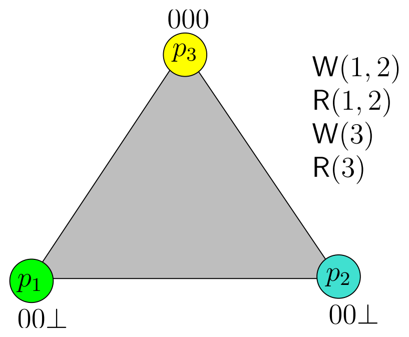
Simulations are an active field in distributed computing, they have been used not only to prove the equivalence of the iterated model and the standard model of distributed computing, but also to extend impossibility results from one model of computation to another, e.g., [BGLR01]. A topological approach to simulations of models of distributed computing can be found in [HR12].

5.3 The geometry of the Iterated model

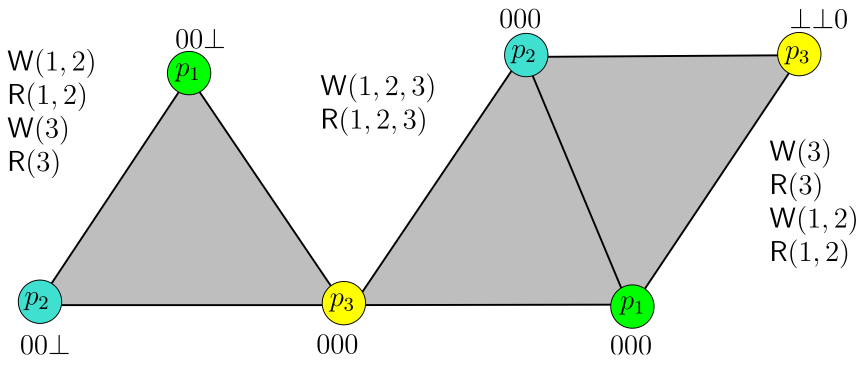
The framework we have described to study distributed algorithms can be given in geometric terms, using simplicial complexes. The well behaved recursive nature of the executions of the protocols in the iterated model allow us to define a sequence of simplicial complexes such that, each member of this sequence represents all the possible ways in which the processes can execute a protocol in a given round number. We assume that the reader is familiar with basic concepts from combinatorial topology [Arm83, Ale98].

5.3.1 The case of 3-process protocols

To introduce the ideas of the geometric approach to distributed algorithms, we begin with an example, let $n = 3$ and suppose processes p_1, p_2 and p_3 execute a protocol $\mathcal{A}$ in the iterated model with their input values all equal to 0. In the first round of $\mathcal{A}$, there are several possible ways for the processes to execute their actions of reading and writing to the shared memory; one possibility is that p_1 and p_2 execute concurrently the basic operations and later, p_3 executes the same steps. This is represented by the round schedule $\mathsf{W}(1,2), \mathsf{R}(1,2), \mathsf{W}(3), \mathsf{R}(3)$. At the end of the round, the local view of the shared memory of p_1 and p_2 only contains the values these processes wrote; they cannot see the value written by process p_3 because they executed faster than p_3, which wrote its information in the shared memory after p_1 and p_2 executed their reads (snapshots) operations[3]. But p_3's local view of the memory contains the data of the three processes, so that p_3 can see all the information written by p_1 and p_2. This global state, which is reachable in $\mathcal{A}$, can be represented by a 2-simplex Δ_2 as shown in the following figure.



Each vertex of Δ_2 is labeled with the process id and the local state of that process (this includes the local view of the memory). Now consider the case where the three processes execute the shared operations concurrently (this is described with the round schedule $\mathsf{W}(1,2,3), \mathsf{R}(1,2,3)$) and the views they have of the memory at the end of the first round. As in the previous case, we represent this state as a 2-simplex Δ_2'.



This time, all the processes can see each other's input values and because p_1 and p_2 have a different local view of the memory, they can distinguish between the state given by Δ_2' and the previous one (represented by Δ_2); but p_3 cannot tell the difference, because it has exactly the same information

[3]This models the possibility that p_3 is just running slower that p_1 and p_2 and also it models the situation in which p_3 crashes and does not take any steps at all.

in both states. This is the reason why the simplexes Δ_2 and Δ'_2 have the vertex with p_3's id as a common face, p_3 cannot distinguish between the two states, because in both of them it has the same local state. In a similar way, the simplex that represents the reachable state in $\mathcal{A}$ obtained by executing the first round of the protocol in the way described by $\mathsf{W}(3), \mathsf{R}(3), \mathsf{W}(1,2), \mathsf{R}(1,2)$, is adjacent to Δ'_2 (this time, p_1 and p_2 cannot distinguish).

We can represent all the possible reachable states in the first round of $\mathcal{A}$ as 2-simplexes and construct a simplicial complex, which we call the *1-round protocol complex* of $\mathcal{A}$ (Figure 5.2 (a)). It is just a subdivided triangle. Each simplex in this complex represents a state S that the system can reach after the processes execute the first round of $\mathcal{A}$ with a round schedule that makes the protocol enter into state S.

In the second round, processes start to execute the protocol with the state they had at the end of the first round, and then all the possible round schedules of the first round can be repeated. If we represent the state that the processes have at the beginning of round two as the corresponding 2-simplex of the 1-round protocol complex of $\mathcal{A}$ (that is, as a reachable state of the first round of the protocol) then using the fact that we are working with an iterated model of computing, we can encode all the possible states of the system at the end of round two as a subdivision of the triangle that represents the state that the processes had at the end of round one. This subdivision is identical to the protocol complex of round one, so that we can describe all the possible reachable states in round two of $\mathcal{A}$ as the simplicial complex of Figure 5.2 (b), this is the *2-round protocol complex of $\mathcal{A}$*. Notice that this complex is also a subdivided triangle, with several subcomplexes isomorphic to the protocol complex of round one. This behavior is going to repeat itself for all subsequent rounds– the *k-round protocol complex* for a protocol in the iterated model for three processes will be a subdivision of the 2-simplex. To obtain the complex of a subsequent round, we just take the complex of the previous round and replace each simplex in it with the complex of Figure 5.2 (a) and we can see that the k-round protocol complex has a simple and elegant recursive structure.

5.3.2 The protocol complex

In general, if we have $n+1$ processes, each possible initial or final local state of a process is modeled as a vertex $v = (i, sm_i)$, a pair consisting of a process id i and the local state sm_i of process p_i. We say that the vertex is *coloured* with the process id. A set of $n+1$ mutually compatible initial or final local states is represented as an n-simplex Δ_n and with this we model a possible system state. Given a protocol $\mathcal{A}$ in the iterated model, where each process p_i receives the input value v_i ($i = 1, \ldots, n+1$), for each round number $r \geqslant 0$ all the possible reachable states in $\mathcal{A}$ at the end of round r are represented as a n dimensional complex, the *r-round protocol complex* (for brevity, we just say the "protocol complex"), $\mathcal{P}_{\mathcal{A}}$, in which each vertex is labeled with a process id and that process state at the end of round r. Thus, each simplex in $\mathcal{P}_{\mathcal{A}}$ corresponds to an equivalence class of executions of $\mathcal{A}$ that "look the same" to the processes at its vertices. It is argued in [BG97] that $\mathcal{P}_{\mathcal{A}}$ is always a subdivided n-simplex.

There is more information about the protocol complex in [HS99]. In fact, the definition we have presented of the r-round protocol complex $\mathcal{P}_{\mathcal{A}}$ (associated with the input values that the processes have when they begin executing the protocol $\mathcal{A}$) is closely related with the notion of a *span* in [HS99, Definition 4.4, page 884]. Herlihy and Shavit define the protocol complex as a bigger geometric structure which depends on all the possible input values of processes and all possible executions of a protocol $\mathcal{A}$. However, for most cases, it is sufficient to work with the definition of protocol complex we have given. We find our definition sufficient for this introductory note and for the rest of this thesis.

The use of topological methods in distributed computing is growing and evolving everyday, if the reader wishes to get to know the most up to date information and newest techniques (at the time of writing this thesis) on this topic, some good references are [HR10a, Her10, HRR12, HKR13]

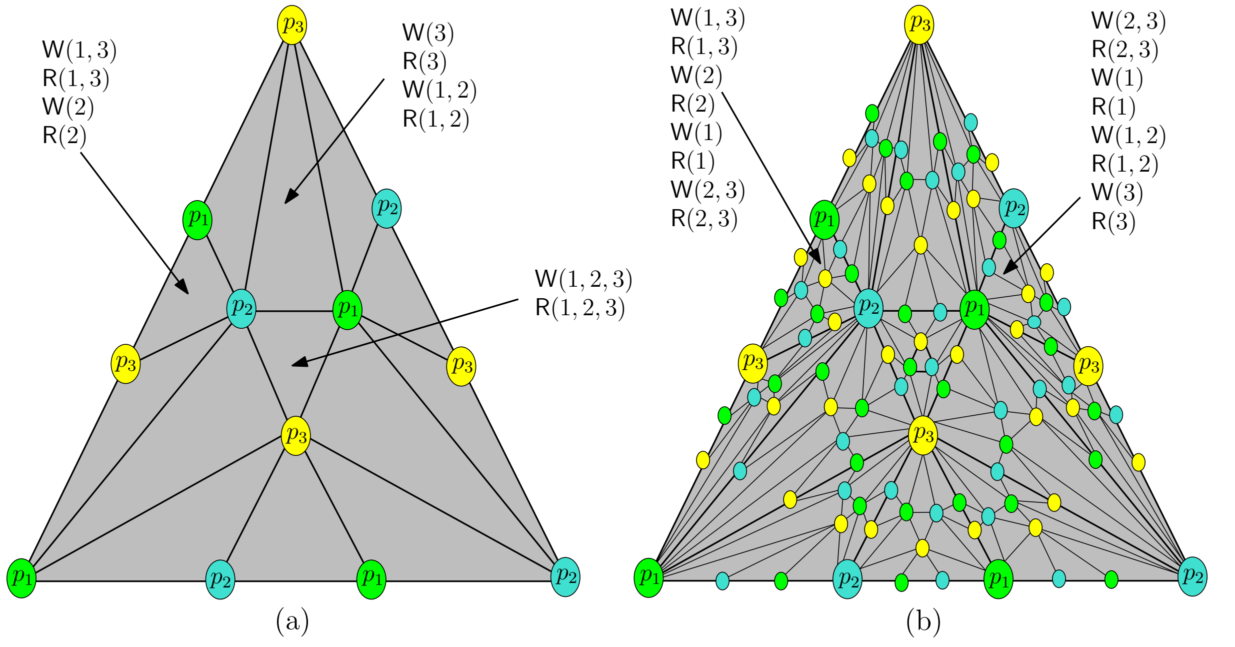


Figure 5.2: The protocol complex for a 3-process protocol in the iterated model after the execution of the first round (a) and the second round (b).

5.4 Consensus and the family of set agreement tasks

In the consensus task for n processes, each process starts executing with a private input value and all processes must execute a finite number of steps to decide a unique output value v that must be an input value of some correct process. The consensus task is fundamental in the theory of distributed computing. Consensus is used to achieve overall system reliability in the presence of faulty processes, and is used in databases for transaction commitment, agreeing on the identity of a leader, state machine replication, and atomic broadcasts. The consensus task is usually described in terms of three requirements: *termination*, *agreement* and *validity*. Each process proposes a value, and each correct process (one that does not crash) has to decide a value (termination), in such a way that there is a single decided value (agreement) and that value is a proposed value (validity).

A seminal result in the theory of distributed computing is the impossibility of solving consensus using a shared-memory that consists of read/write registers with asynchronous processes even if only one process may crash [FLP85, LAA87]. Given the huge practical importance of consensus, many situations have been studied to ameliorate the implications of this impossibility result, from stronger models of distributed computing, to weaker versions of consensus. It must be noticed that modern multi-processor systems include communication objects that are more powerful than read/write registers.

5.4.1 The set agreement task

The (n,k)-set agreement task, where n is the number of processes and $k < n$, is one of the most important generalization of consensus. This task was proposed the first time in [Cha93] and it has been fundamental in the study of distributed computing.

The (n,k)-set agreement task. In this task, every process starts with some initial input value taken from a set I of input values[4] and must output a value such that:

- Termination: Each process must eventually output some value.
- k-Agreement: The set of output values from all processes must be of size at most k.
- Validity: If some process outputs v, then v is the initial input of some process.

We can define formally the *consensus* task for n processes as the $(n,1)$-set agreement task and the 1-Agreement condition is simply called Agreement. As we have said, before the set agreement task was defined, it was well known that the consensus task is not solvable in the presence of even only one faulty process [FLP85, LAA87]. Almost all consensus impossibility results use graph connectivity arguments. But since the (n,k)-set agreement task was conceived, it was an open question whether it could be solved in the wait-free shared memory model (with the parameters $2 \leqslant k < n$), connectivity of graphs was not enough to give an impossibility proof. It was until 1993 when three independent teams [BG93a, SZ93, HS93] showed that there is no wait-free protocol that can solve set agreement.

Theorem 5.4.1 ([BG93a, SZ93, HS93])**.** *For $1 \leqslant k < n$, the (n,k)-set agreement task has no wait-free read/write solution in the iterated shared memory model.*

The proof of Theorem 5.4.1 uses the topological framework given earlier to define the iterated model and very popular results from topology, like Sperner's Lemma. As the iterated model is equivalent to the usual standard wait-free shared memory model, Theorem 5.4.1 implies the impossibility to solve the set agreement task in the standard model. It is worth noticing that the set agreement task is not the only distributed task in which topological techniques have been useful. Other examples are musical benches [GR05], renaming [HS99, CnR08, CnHR12, CnR12b], loop agreement [HR03] and the weak symmetry breaking task [CnR12a] among others. The consensus impossibility results of [FLP85, LAA87] can be seen as special cases of the topological impossibility proof of set agreement.

5.4.2 The impossibility of consensus in the iterated model

Theorem 5.4.1 tell us that the consensus task cannot be solved with a protocol in the iterated model. We now present a variant of the proof of this impossibility result. We will show that using simple connectivity arguments and some special properties of global states of a system, we can show with a well structured proof that there is no protocol in the iterated model that can implement the consensus task for $n \geqslant 2$ processes.

[4]It is customary to ask that $|I| \geqslant n$, although for consensus, which is the $(n,1)$-set agreement task, we only ask that $|I| \geqslant 2$.

A 2-process example

To illustrate the basic ideas and the geometry behind the proof of the impossibility of consensus in the iterated model, let us give an example with $n = 2$ processes, from which a generalization to any $n \geqslant 2$ is straightforward. Suppose then that $\Pi = \{p_1, p_2\}$ and $\mathcal{A}$ is a protocol that solves the consensus task for 2 processes, without loss, assume that the input values of the processes are their own ids, so that p_1 has input value 1 and p_2 input value 2, this is represented by an initial global state I. If the processes execute the protocol $\mathcal{A}$ for one round, then we can see that the 1-round protocol complex of $\mathcal{A}$ is the 1-dimensional simplicial complex $\mathcal{P}_1$ at the top of Figure 5.3. $\mathcal{P}_1$ represents all three possible executions of $\mathcal{A}$ in the first round, each of the 1-simplexes of $\mathcal{P}_1$ represents a reachable state in $\mathcal{A}$ in the first iteration. All possible one round executions of $\mathcal{A}$ are given by the round schedules $\pi_1\colon \mathsf{W}(1), \mathsf{R}(1), \mathsf{W}(2), \mathsf{R}(2)$; $\pi_*\colon \mathsf{W}(1,2), \mathsf{R}(1,2)$ and $\pi_2\colon \mathsf{W}(2), \mathsf{R}(2), \mathsf{W}(1), \mathsf{R}(1)$. π_1 represents an execution of $\mathcal{A}$ in which p_1 executes the write and read operations in solo and p_2 awakes later and executes the same operations. The execution given by π_2 is symmetric to π_1, we just interchange p_1 and p_2. The round schedule π_* represents an execution of $\mathcal{A}$ in which both p_1 and p_2 execute concurrently the write and read operations of $\mathcal{A}$. Thus, the three reachable states represented by $\mathcal{P}_1$ are $I_1 = I \cdot \pi_1, I_* = I \cdot \pi_*$ and $I_2 = I \cdot \pi_2$. The connectivity of the complex $\mathcal{P}_1$ tell us that $I_1 \overset{2}{\sim} I_* \overset{1}{\sim} I_2$. This is easy to show, p_2 cannot distinguish between I_1 and I_*, because in both states, p_2 sees the input value of p_1 and its own input; similarly, p_1 cannot distinguish between I_* and I_2, because it can see the input values of both processes in the two states.

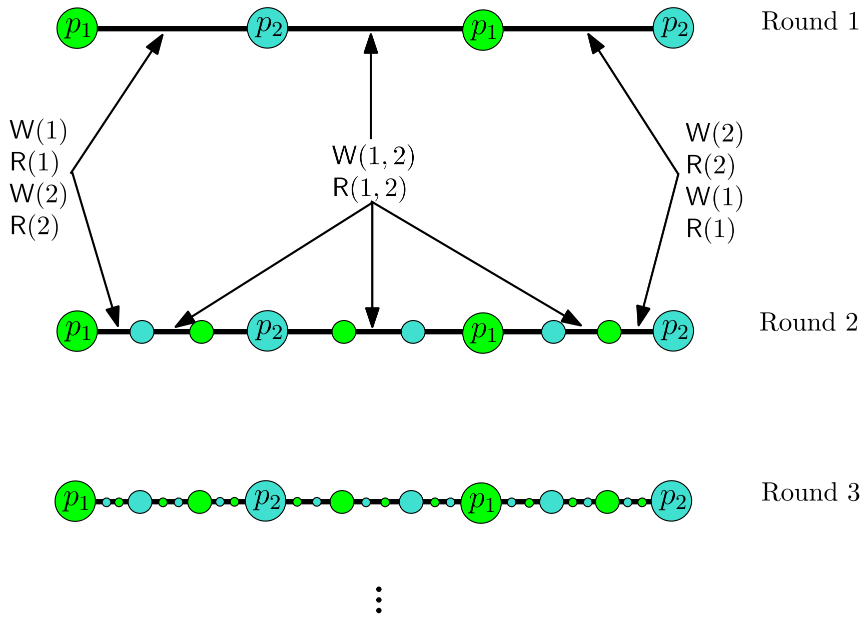


Figure 5.3: The k-round protocol complex for 2 processes and $k \in \{1,2,3\}$.

If the processes halt and decide a unique output value in the first round of $\mathcal{A}$, then in the state I_1, p_1 decides 1, because it executes in solo and it never sees p_2's input value and by the Agreement property of consensus, p_2 must decide 1 too. Now, as process p_2 cannot distinguish between I_1 and I_* and p_2 decides 1 in the state I_1, p_2 decides the same output value in I_* and because of the Agreement condition, p_1 must decide 1 in I_*. Everything seems to be okay until this point, but what happens in the state I_2 ? We know that process p_2 executed faster that p_1, so that the output value decided by p_2 is 2 and (again) by the Agreement property of consensus, p_1 decides 2 in I_2 On the other hand, we have that p_1 decided the output value 1 in the state I_* and as $I_* \overset{1}{\sim} I_2$, p_1 has to decide 1 in I_2, but we just have argued that p_1 outputs 2 as its decision in I_2, thus we have reached a contradiction. We conclude that the processes must execute at least one more round of $\mathcal{A}$.

If we now suppose that p_1, p_2 execute $\mathcal{A}$ until the end of round two and decide, then we can build a similar contradiction. The 2-round protocol complex $\mathcal{P}_2$ of $\mathcal{A}$ is depicted in the middle of Figure 5.3. Here we can see the recursive nature of the protocol complex of protocols in the iterated model. $\mathcal{P}_2$ is obtained from $\mathcal{P}_1$ simply by replacing each 1-simplex of $\mathcal{P}_1$ by the complete complex $\mathcal{P}_1$. Roughly speaking, each 1-simplex of $\mathcal{P}_2$ represents an execution of $\mathcal{A}$ in which the processes start executing round two of $\mathcal{A}$ after finishing the first iteration in the global state I_l ($l \in \{1, 2, *\}$) and executing round 2 in the way specified by one of the round schedules π_1, π_2 and π_*. In summary, the simplexes of $\mathcal{P}_2$ represent reachable states in the second iteration of $\mathcal{A}$ which have the form

$$I_l \cdot \pi_s \qquad l, s \in \{1, 2, *\}.$$

Consider now the states $I_1 \cdot \pi_1$ and $I_2 \cdot \pi_2$. The first state represents and execution of $\mathcal{A}$ up to round two in which, process p_1 executes in solo the write and read operations in round 1 and 2 of $\mathcal{A}$; the second state represents a similar execution of $\mathcal{A}$ for p_2. The geometry of the complex $\mathcal{P}_2$ says that these two states are connected (this can be proven by an argument very similar to the one that shows that $I_1 \overset{2}{\sim} I_* \overset{1}{\sim} I_2$) and this can be used to reach a contradiction in the same way that we did for the round one case. Using the recursive nature of the iterated model and the protocol complex of $\mathcal{A}$, we can reach the same contradiction in round three (see bottom of Figure 5.3) and all subsequent rounds of $\mathcal{A}$. A reachable state of an execution of $\mathcal{A}$ in which process p_1 executes every round in solo and p_2 is delayed indefinitely, can be connected with a reachable state of an execution of $\mathcal{A}$ where p_2 executes faster that p_1 forever.

The impossibility proof

We are ready to give the formal proof of the impossibility of consensus in the iterated model, generalizing the intuitive ideas given in the previous example. We begin first by introducing some definitions regarding consensus protocols.

Definition 5.4.2. Let $\mathcal{A}$ be a protocol (in any model) that implements consensus, v a valid input value for consensus and S a state of the system. We say that S is *v-valent* if in every execution of $\mathcal{A}$ starting from S, there exists a process in Π that outputs v. S is *univalent* if in every execution of $\mathcal{A}$ starting from S, processes always outputs the same value. If S is not univalent, then S is *bivalent*.

Fix $i \in \overline{n}$ and define the round schedules π_i, π_* and $\pi_{\overline{n}-i}$ as

(π_i) $\mathsf{W}(i), \mathsf{R}(i), \mathsf{W}(\overline{n}-i), \mathsf{R}(\overline{n}-i)$,

(π_*) $\mathsf{W}(\overline{n}), \mathsf{R}(\overline{n})$,

($\pi_{\overline{n}-i}$) $\mathsf{W}(\overline{n}-i), \mathsf{R}(\overline{n}-i), \mathsf{W}(i), \mathsf{R}(i)$.

Lemma 5.4.3. *Any two initial states of a protocol for consensus are connected.*

Proof. Let S, P be two initial states. If S and P differ only in the initial value $input_i$ of a single process p_i, then S and P are adjacent (Only p_i can distinguish between the two states, the rest of the processes have the same initial values). In the case S and P differ in more that one initial value, they can be connected by a path of initial states $S = S_1 \sim \cdots \sim S_q = P$ such that S_j, S_{j+1} differ only in the initial value of some process (we obtain S_{j+1} from S_j by changing the input value of some process p_i, the result is a valid input of the consensus task), hence they are adjacent. In summary, S and P are connected. □

Lemma 5.4.4. *Let $n \geqslant 2, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the iterated model and S is any reachable state in some round $r \geqslant 0$ of $\mathcal{A}$, then there exists the path*

$$S \cdot \pi_i \overset{\overline{n}-i}{\sim} S \cdot \pi_* \overset{i}{\sim} S \cdot \pi_{\overline{n}-i}, \tag{5.1}$$

of connected reachable states in round $r + 1$ of $\mathcal{A}$.

Proof. Let S be a reachable state in $\mathcal{A}$ and $i \in \overline{n}$. We show that the states $S \cdot \pi_i$ and $S \cdot \pi_*$ are connected. In the first state, process p_i first executed in solo and it could not see any other process, but the rest of the processes executed concurrently, thus they can see the information from all processes after executing the snapshot operation. For the state $S \cdot \pi_*$, all processes executed concurrently, so that they can see each other's data after the snapshot, therefore all processes with ids in the set $\overline{n} - i$ have the same local state in $S \cdot \pi_i$ and $S \cdot \pi_*$, thus they cannot distinguish between these two global states. A similar argument proves that p_i cannot distinguish between $S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$. It follows that the path $\boxed{5.1}$ of connected reachable states in $\mathcal{A}$ exists. □

Lemma 5.4.5. *Let $n \geqslant 2, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the iterated model and there exists a path*

$$S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l \qquad (l \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that for all j with $1 \leqslant j \leqslant l$, $X_j = \overline{n} - i$ or $X_j = \{i\}$. Then there exists a path

$$Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_s}{\sim} Q_s \qquad (s \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that

(1) Every state Q_t is a successor state of some S_j.

(2) For all m with $1 \leqslant m \leqslant s$, Z_m is such that $Z_m = \overline{n} - i$ or $Z_m = \{i\}$.

Proof. We use induction on l, the round schedules $\pi_i, \pi_*, \pi_{\overline{n}-i}$ and Lemma 5.4.4 to construct the path $Q_1, \ldots, Q_s$ of connected states with the desired properties. For the base case, consider the states $S_0 \overset{X_1}{\sim} S_1$ where $X_1 = \{i\} \vee X_1 = \overline{n} - i$. It is easy to see that the state $S_0 \cdot \rho$ is adjacent to the state $S_1 \cdot \rho$, where ρ is π_i if $X_1 = \{i\}$ and is $\pi_{\overline{n}-i}$ if $X_1 = \overline{n} - i$. Assume that we have build the path

$$Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_{s'}}{\sim} Q_{s'},$$

of connected successor states of $S_1, \ldots, S_q$ $(1 \leqslant q < l)$ with $Z_m = \overline{n} - i$ or $Z_m = \{i\}$ for all $m \leqslant s'$. Each state Q_m is of the form $Q_m = S_j \cdot \alpha$, where $\alpha \in \{\pi_i, \pi_*, \pi_{\overline{n}-i}\}$. We now show how to connect $Q_{s'}$ (a successor state of S_q) with a successor state of S_{q+1}. Let X_{q+1} be the set of processes that cannot distinguish between S_q and S_{q+1}. We have to deal with cases.

Case 1. $Q_{s'} = S_q \cdot \pi_i$. If $X_{q+1} = \{i\}$, then we can connect $S_q \cdot \pi_i$ with $S_{q+1} \cdot \pi_i$ (p_i cannot distinguish between these two states). Now, if $X_{q+1} = \overline{n} - i$, using Lemma 5.4.4 we can build the path

$$S_q \cdot \pi_i \overset{\overline{n}-i}{\sim} S_q \cdot \pi_* \overset{i}{\sim} S_q \cdot \pi_{\overline{n}-i} \overset{\overline{n}-i}{\sim} S_{q+1} \cdot \pi_{\overline{n}-i}.$$

Case 2. $Q_{s'} = S_q \cdot \pi_*$. Whether $X_{q+1} = \{i\}$ or $X_{q+1} = \overline{n} - i$ we have the path of connected states

$$S_q \cdot \pi_* \overset{X_{q+1}}{\sim} S_q \cdot \pi_{X_{q+1}} \overset{X_{q+1}}{\sim} S_{q+1} \cdot \pi_{X_{q+1}}.$$

Case 3. $Q_{s'} = S_q \cdot \pi_{\overline{n}-i}$. The argument here is very similar to Case 1, so we omit it.

We have built with induction the path of connected states $Q_1, \ldots, Q_s$ from the path $S_1, \ldots, S_l$ satisfying the demanded properties. The result follows. □

Lemma 5.4.6. *Let $n \geqslant 2, i \in \overline{n}$. Suppose that $\mathcal{A}$ is a protocol in the iterated model that solves the consensus task and that $S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l$ ($l \geqslant 1$) is a path of connected reachable states in $\mathcal{A}$ that satisfies the hypothesis of Lemma 5.4.5 and also assume that S_0 is a v-valent state. Then S_l is v-valent.*

Proof. It is enough to prove the lemma for $l = 1$, as the general case follows easily from this case. Suppose (without loss of generality) that $S = S_0$ is v-valent and that $S' = S_l$ is v'-valent ($v \neq v'$). Since S is v-valent, in every execution starting from S, there is at least one process that outputs v and by the Agreement condition of consensus, all processes must output v as the consensus value; the same is true for S', replacing v with v'. Let r be the round number of both S and S' (this makes sense, because S and S' are adjacent, thus they must have the same round number, which is part of the local state of each process), then combining Lemma 5.4.5 with an inductive argument, we can find for all $m \geqslant r$ and any two r-round partial executions

$$R_0, \pi_1, \ldots, \pi_r, R_r = S \quad \text{and} \quad P_0, \pi'_1, \ldots, \pi'_r, P_r = S',$$

that end in S and S' respectively, m-round partial executions

$$R_0, \pi_1, \ldots, \pi_r, R_r, \pi_{r+1}, \ldots, \pi_m, R_m \quad \text{and} \quad P_0, \pi'_1, \ldots, \pi'_r, P_r, \pi'_{r+1}, \ldots, \pi'_m, P_m,$$

such that R_m and P_m are adjacent states for all $m \geqslant r$. As the protocol solves the consensus task, there must exists a k such that R_u and P_u are decision states for all $u \geqslant k$. Without loss, we can assume that $k \geqslant r$. Since R_u is a successor state of S, which is a v-valent state, all processes decide v in R_u; simillary, as P_u is a successor of S' (a v'-valent state), processes must decide v' in P_u. Let X be the set of ids of processes that cannot distinguish between R_u and P_u. Then, for each $j \in X$ the local state of process p_j in P_u and R_u must be the same and this includes its output component. But this is a contradiction because in R_u, p_j decides v while in the state P_u, p_j decides v'. We conclude that S' must be v-valent, such as S. □

Theorem 5.4.7. *For $n \geqslant 2$, there is no protocol in the iterated model to solve consensus for n processes.*

Proof. Assume that there exists a protocol $\mathcal{A}$ for consensus in the iterated model and (without loss of generality) suppose that 0,1 are two valid input values. Let i be any process id and let O, U be the initial states in which all processes have as input values 0s and 1s respectively. Clearly, O is a 0-valent state and U is a 1-valent state. Let OU be the initial state in which p_i has input value

0 and all other processes have input value 1. Then in the first round of $\mathcal{A}$, we have the following path of connected reachable states in $\mathcal{A}$:

$$O \cdot \pi_i \overset{i}{\sim} OU \cdot \pi_i \overset{\overline{n}-i}{\sim} OU \cdot \pi_* \overset{i}{\sim} OU \cdot \pi_{\overline{n}-i} \overset{\overline{n}-i}{\sim} U \cdot \pi_{\overline{n}-i}.$$

Because $O \cdot \pi_i$ is 0-valent, Lemma 5.4.6 tell us that the state $U \cdot \pi_{\overline{n}-i}$ is also 0-valent. But this contradicts the fact that $U \cdot \pi_{\overline{n}-i}$ is a 1-valent state. Therefore no such protocol $\mathcal{A}$ can exists. $\square$

By Theorem 5.2.3, the consensus task for n processes is impossible to solve with a protocol in the standard model. So, both the set agreement and the consensus tasks are impossible to solve with a shared memory. To implement these tasks, more powerful objects are needed. In the next chapters, we define what we call an extension of the iterated model, in which we add another kind of shared objects, besides shared memories, and we will study the computational power of this extension using set agreement and consensus.

6 An enrichment of the iterated model

In the last chapter, we have introduce a basic framework to study distributed computing and with it, we defined the basic iterated model, in which distributed protocols (algorithms) are executed by the processes in a round-based pattern and the processes perform the write and read operations in the shared memory with a well defined order and each operation is executed only once. Also, the shared memory is renewed in each iteration, with the help of the snapshot objects. We also defined the (n, k)-set agreement task and the consensus task for n processes and proved the impossibility to solve consensus with a protocol in the iterated model.

In this chapter, we add more powerful shared objects to the iterated model, and we investigate the solvability of the set agreement and consensus task in this extended model. To define shared objects which are more powerful that a shared memory, we will use a technique very popular in standard computability theory: *Black boxes*. We will take a distributed task, which is unsolvable with shared memory and assume that we have a black box that can solve it. Then we define shared objects that satisfy the properties of the unsolvable task and add them to the basic iterated model. This kind of technique has been used in previous research to define extensions of the iterated model and compare the computational power of the set agreement task and the renaming task [GRH06, CnIRR12]. In our case, the task that we will use to define new shared objects stronger that the shared memory, is called the *safe-consensus* task [AGL09], another variant of consensus.

In this new model, we will study the topology of the protocols in the extended model. We will see that this topology is in general very different from the topology of protocols in the basic iterated model (see Section 5.3), specifically, we will show with some graphical examples how the protocol complexes of the protocols in the extended model can be disconnected, while the protocol complexes of the protocols in the basic iterated model are always connected (in fact, they are contractible). All this will affect the computational power obtained in the new model with the aid of the new shared objects.

The outline of the chapter is the following. Section 6.1 defines the *general extended iterated model*, which is a modification of the basic iterated model to allow the processes to access objects more powerful than the shared memory and with this model, we define the distributed model of interest for this thesis: the *iterated model with safe-consensus*. In Section 6.2, we study a restriction of the iterated model with safe-consensus, we call it the *IFSC (Iterated with Full Safe-Consensus) model* [CR12]. We use this model to show how the safe-consensus objects affect the protocol complexes of the protocols of the iterated model with safe-consensus. In Section 6.3, we show that the $(n, n-1)$-set agreement task can be implement in the IFSC model, but the consensus task cannot be implemented. Throughout the chapter, we use many figures and examples to help the reader understand the concepts and results.

6.1 A new iterated model of distributed computing

In this section, we define an extension of the basic iterated model of distributed computing. We define a new model in which, the processes proceed to solve a task by iterations and besides being able to use a shared memory (a sequence of snapshot objects), they can use other type of shared objects. Our definition of the new extend iterated model is general, almost any kind of shared object can be used. To retain the ability of obtain interesting results (e.g., impossibility results), we will impose some restrictions to the protocols of the extended iterated model (besides making progress in rounds), these restrictions are very natural and are basically extensions of the restrictions of the iterated model. Later, we will define an instantiation of the extended iterated model, using shared objects which are defined on a variant of consensus: the *safe-consensus* task [AGL09] and from that moment to the end of this thesis, the extended iterated model using safe-consensus objects will be the main model to be investigated in this thesis.

6.1.1 The general extended iterated model

To define our new general model, we begin by extending some basic definition given in Section 5.1. The first definition of process, is the same. We begin with our new definition of a system.

Definition 6.1.1. An *extended system* is a triplet (Π, SM, T), where Π is a set of processes, SM is a set of shared registers and T is a set of *shared objects*. Each shared object $T_i \in \mathsf{T}$ has a domain D_i of input values, and a domain D'_i of output values. T_i provides a unique operation, $\mathsf{T}_i.exec(d)$, that receives an input value $d \in D_i$ and returns instantaneously an output value $d' \in D'_i$.

For simplicity, we omit the word "extended" when we refer to an extended system. We assume that for any system (Π, SM, T), the shared memory is structured as an infinite array of snapshot objects and the set T is an infinite array of shared objects. Also, The *exec* operation of each member of T can be used at most once by each process that invokes it.

The concepts of events, round schedules, protocols, executions and other miscellaneous definitions given in Section 5.1 remain the same. Of course, we need to add a new type of event to denote a shared object invocation done by some set of processes. In Section 6.1.2, when we introduce the *safe-consensus* task of [AGL09], we will add an explicit new event for the invocation of shared objects which are based on the properties of the safe-consensus task.

Definition 6.1.2. Let (Π, SM, T) be a system and $\mathcal{A}$ the induced protocol. We say that $\mathcal{A}$ is a *protocol in the extended iterated model of distributed computing*, if $\mathcal{A}$ can be specified in the form described by Figure 6.1.

As stated in Definition 6.1.2, Figure 6.1 shows the general form of the protocols that we investigate in this thesis, that is, protocols in the extended iterated model. We now give an intuitive explanation of the pseudocode in the Figure, to describe the details of how the processes use the shared memory and the array of shared objects. Initially, r is zero and sm is assigned the contents of the readonly variable $input$, which contains the input value for process p_i; all other variables are initialized to $\perp$. In each iteration, p_i increments by one the loop counter r, accesses the current shared memory array $SM\,[r]$, writing all the information it has stored in sm and val, and then p_i decides which shared object it is going to invoke by executing a deterministic function h that returns an index l, based on the information p_i passes on to h; then p_i invokes the shared object $\mathsf{T}\,[l]$ with some value v (which depends on p_i's local state) as parameter and stores the output value in the local variable *val*. After this, p_i takes a snapshot of the shared array (which contains data from p_i and possibly from other running processes) and finally, p_i checks if *dec* is equal to $\perp$, if so, then with all its information, it enters into a deterministic function δ to determine if it must return a valid output value or $\perp$, which is stored in *dec*. Once a non-$\perp$ value is assigned to *dec*, it is never overwritten. The deterministic function h is the *object selector*.

```
(1)  init $r \leftarrow 0; sm \leftarrow input; dec \leftarrow \perp; val \leftarrow \perp;$

(2)  loop forever
(3)      $r \leftarrow r + 1;$
(4)      $SM[r]$.update$(sm, val);$
(5)      $val \leftarrow \mathsf{T}[h(\langle r, id, sm, val\rangle)].exec(v);$
(6)      $sm \leftarrow SM[r]$.scan$();$

(7)      if $(dec = \perp)$ then
(8)          $dec \leftarrow \delta(sm, val);$
(9)      end if
(10) end loop
```

Figure 6.1: General form of a protocol in the extended iterated model.

Shared objects represented as combinatorial sets

The main result of the second part of this thesis, presented in Chapter 9, is about counting the ways in which the processes can use the shared objects of the array T, in all possible executions of a protocol in the extended iterated model. We now introduce some combinatorial definitions which will help us represent shared objects and the specific way in which the processes can invoke these shared objects.

Definition 6.1.3. Let (Π, SM, T) be a system and $\mathcal{A}$ the induced protocol in the extended iterated model. We define for each $m \leqslant n$ the set $\Gamma_{\mathcal{A}}(n,m) \subseteq \mathcal{P}(\overline{n})$ as follows: $b = \{i_1, \ldots, i_m\} \in \Gamma_{\mathcal{A}}(n,m)$ if and only in some iteration of the protocol $\mathcal{A}$, only the processes $p_{i_1}, \ldots, p_{i_m}$ invoke a shared object of the array T.

Roughly speaking, each $c \in \Gamma_{\mathcal{A}}(n,m)$ represents a set of processes which together can invoke shared objects in $\mathcal{A}$. For example, if $m = 3$ and $c = \{i,j,k\} \in \Gamma_{\mathcal{A}}(n,3)$, then in at least one round of $\mathcal{A}$, processes p_i, p_j and p_k invoke an object $\mathsf{T}[l]$ (for some index l) and if in other iteration or perhaps another execution of $\mathcal{A}$, these processes invoke another shared object $\mathsf{T}[l']$ ($l \neq l'$) in the same way, then these two invocation are represented by the same set $c \in \Gamma_{\mathcal{A}}(n,3)$, that is, shared objects invoked by the same set of processes are considered as the same element of $\Gamma_{\mathcal{A}}(n,m)$. In short, repetitions do not count.

Definition 6.1.4. Let $\mathcal{A}$ be a protocol in the extended iterated model. A set $b \in \Gamma_{\mathcal{A}}(n,m)$ is called an *m-box* or simply a *box*. Let the set $\Gamma_{\mathcal{A}}(n)$ and the quantities $\nu_{\mathcal{A}}(n,m)$, $\nu_{\mathcal{A}}(n)$ be defined as follows:

$\Gamma_{\mathcal{A}}(n) = \bigcup_{m=2}^{n} \Gamma_{\mathcal{A}}(n,m);$

$\nu_{\mathcal{A}}(n,m) = |\Gamma_{\mathcal{A}}(n,m)|;$

$\nu_{\mathcal{A}}(n) = \sum_{m=2}^{n} \nu_{\mathcal{A}}(n, m)$.

Extended global states

We need to give a new Definition of global state, so that this definition can describe the information obtained from the shared objects of the array T.

Definition 6.1.5. Let (Π, SM, T) be a system. An *extended state* is a vector

$$S = \langle s_1, \ldots, s_n; SM; b_1, \ldots, b_q; o_1, \ldots, o_q \rangle,$$

where s_i is the local state of process p_i, SM is the state of the shared memory, $b_1, \ldots, b_q$ are the boxes that specify the way in which the processes invoked some shared objects from T to enter the local states s_i ($i = 1, \ldots, n$) and for $j = 1, \ldots, q$, o_j is the output value[1] of the shared object represented by the box b_j.

Of course, the information represented by the boxes $b_1, \ldots, b_q$ and the output values $o_1, \ldots, o_q$ of the shared objects represented by the boxes are included implicitly in the local states of the processes. However, for practical purposes, it is very convenient to specify the boxes and the output values of the shared objects explicitly in the definition of extended global state. As we did in Chapter 5 for global states, for simplicity, we will refer to an extended state of a system simply as a state. With this new concept of global state, the definition of initial state changes too: An *initial state* is a state in which every local state is an initial local state, all registers in the shared memory are set to $\perp$, the invocation specification is empty and the set of output values is empty.

To begin working with the extend iterated model, one last definition is needed, this formalizes the idea of using a task to implement another task.

Definition 6.1.6. Let S, T be distributed tasks. We say that *S implements T*, if there exists a protocol $\mathcal{A}_S$ (in the extended model) that solves T and uses shared objects that are instances of task S.

6.1.2 The extended iterated model with safe-consensus

With the definition of the general extended iterated model of distributed computing at hand, we only need one more thing to introduce the new model that we want to study for the rest of this thesis. We now define another variant of the consensus task, which is called the *safe-consensus task* [AGL09]. This task is the basis to build the shared objects that we need to define an instantiation of the general extended iterated model. First things first, here is the definition of safe-consensus.

The safe-consensus task. In this task, every process starts with some initial input value taken from a set I and must output a value such that Termination and Agreement of consensus are satisfied and also:

- Safe-Validity: (1) If a process p_i starts invoking the task and outputs before any other process starts executing the task, then the task's output is p_i's proposed input value. (2) Otherwise, if two or more processes initially access the safe-consensus task concurrently, then the task can return any value from a countable set V such that I is a proper subset of V.

[1]For the purposes of this thesis, we assume that the tasks that are used as shared objects in our protocols, return a unique output value. It is not hard to generalize this definition to include shared objects that can return different output values to the processes.

We ilustrate in Figure 6.2 the Safe-Validity condition of safe-consensus, with an example of a three process execution. In part (1), process p_3 begins invoking the safe-consensus task and it outputs before p_1 and p_2 start some computation, thus the output value of all the processes must be p_3's input value, accordingly to (1) of the Safe-Validity property. In part (2), p_2 and p_3 execute concurrently and they invoke the safe-consensus task, therefore the output value of the task can be arbitrary, as it is specified in (2) of Safe-Validity condition. The value of the consensus output α can be u, v, w or any other value different from the previous three values and the later case can only happen if there is a concurrent access to the task.

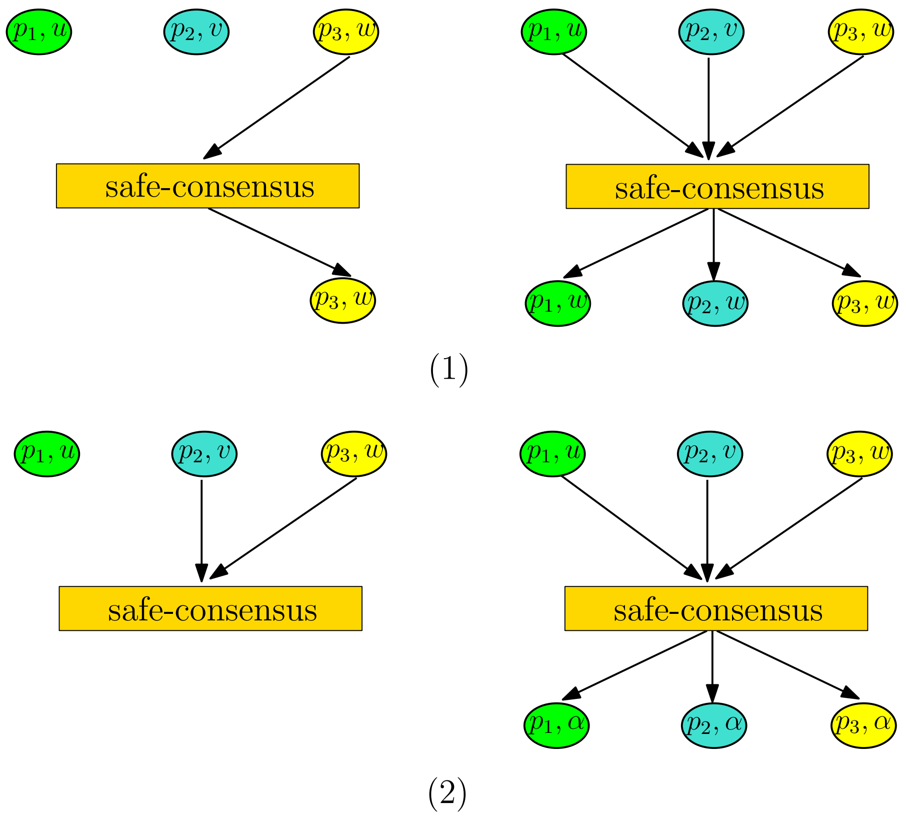


Figure 6.2: A graphical example of the Safe-Validity condition of the safe-consensus task.

The safe-consensus task is the result of weakening the validity condition of consensus. It was first proposed by Yehuda, Gafni and Lieber [AGL09]; they use it to show that the g-tight-group-renaming task [AGL$^+$08] is as powerful as consensus for g processes. Clearly, n-consensus can implement safe-consensus for n processes. It is shown in [AGL09] that consensus of n processes can be implemented with read/write registers and safe-consensus objects. Two protocols are constructed in [AGL09] to solve consensus with safe-consensus, the first is somewhat simpler but it requires $\Theta(2^n)$ safe-consensus black boxes; the second protocol is more involved but it only needs $\Theta(n^2)$ safe-consensus shared objects to implement consensus. This says that safe-consensus is as powerful as consensus, as long as task solvability is concerned. Therefore by the results of [AGL09] and Theorem 5.4.7, it is impossible to solve the safe-consensus task for n processes in the basic iterated model and the same is true for the standard shared memory model of distributed computing.

We can now define new objects to extend the basic iterated model of Chapter 5.

Definition 6.1.7. A *safe-consensus object* is a shared object that can be invoked by any number of processes. The object receives an input value from each process that invokes it, and returns to all the processes an output value that satisfies the Agreement and Validity condition of the safe-consensus task.

In other words, a safe-consensus object is like a "black box" that the processes can use to solve instances of the safe-consensus task. The method of using distributed tasks as black boxes inside protocols is a standard way to study the relative computational power of distributed tasks (i.e. if one task is weaker than another, see [GRH06, AGL09, CnIRR12]). By the arguments given in the paragraph just before Definition 6.1.7, safe-consensus shared objects are primitives more powerful than the read/write shared memory registers.

Definition 6.1.8. Let (Π, SM, SC) be a system, where the shared objects in the array SC are safe-consensus objects. A protocol $\mathcal{A}$ in the extended iterated model that uses the safe-consensus objects of SC is called a *protocol in the iterated model with safe-consensus objects*.

In the iterated model with safe-consensus, we denote the event of a call to a safe-consensus object by the symbol S.

6.2 The Iterated model with Full Safe-consensus objects

In general, studying a distributed model with shared memory and shared objects is difficult. If the shared objects of the array T in Figure 6.1 are equivalent to read/write registers in task solvability, then the extended model is not a new model, it is equivalent to the ordinary standard model of distributed computing. But if the shared objects in T are more powerful that shared memory (like the safe-consensus shared objects) then we have indeed a new extended model, but we may lose some of the nice features of the basic iterated model, like its repeated recursive structure in the protocol complex. Loosing the simplicity of the iterated model is a necessary condition if we want to obtain a distributed model in which the protocols can solve a wider range of tasks and this can be seen in a geometric way using the protocol complex. The main obstacle which does not allow a protocol in the iterated model to solve set agreement tasks is the high connectivity of its protocol complex. Thus, if a protocol in a different model can implement a set agreement task, then we must expect the connectivity of the protocol complex to diminish. In this section, we introduce a distributed model that is a simple and elegant example that shows our arguments to be true. Also, this model will be very useful to introduce some techniques that we will use in later chapters, where we present our main results.

6.2.1 The IFSC model

We begin our research of the iterated model with safe-consensus objects with a distributed model that lies somewhere between the basic iterated model and the iterated model with safe-consensus. Besides the restrictions imposed to protocols in the extended iterated model given in Section 6.1.1, we also assume two additional properties:

E1 In each iteration, all the processes invoke the same safe-consensus object, that is, there is only one safe-consensus object in each round and it is invoked by all processes.

E2 The input value that processes feed to the safe-consensus objects is their own[2] ids.

We call this model of computation the *Iterated shared memory with full safe-consensus objects* (IFSC) model (Figure 6.3). It is not hard to see that we can assume E2 true without loss of generality. A safe-consensus shared object of arbitrary input domain, can be emulated with a safe-consensus object of input domain $\overline{n}$ and shared memory.

```
(1)  init $r \leftarrow 0; sm \leftarrow input; dec \leftarrow \perp; val \leftarrow \perp;$

(2)  loop forever
(3)      $r \leftarrow r + 1;$
(4)      $SM[r]$.update$(sm, val);$
(5)      $val \leftarrow safe\text{-}consensus[r].exec(id);$
(6)      $sm \leftarrow SM[r]$.scan$();$

(7)      if $(dec = \perp)$ then
(8)          $dec \leftarrow \delta(sm, val);$
(9)      end if
(10) end loop
```

Figure 6.3: General form of a protocol in the IFSC model

6.2.2 The protocol complex with safe-consensus

What happens to the protocol complex in the iterated model with safe-consensus ? Are the topological properties unchanged ? In order to answer this question, we use the IFSC model. The first thing that we must do, is to formally define the protocol complex for protocols in the new IFSC model. If $\mathcal{A}$ is a protocol in the IFSC model for n processes, the *protocol complex* $\mathcal{S}_{\mathcal{A}}$ of $\mathcal{A}$ is defined in the same way as we defined protocol complexes in Section 5.3, the only thing that changes is the view associated to a vertex (which represents a local state of some process). A vertex of $\mathcal{S}_{\mathcal{A}}$ is a tuple

$$v = (i, sm_i, val),$$

where i is the id of process p_i, sm_i is the local state of p_i and val is the return value of the safe-consensus object invoked by all processes. In Figure 6.4 we have a drawing of a part of the protocol complex $\mathcal{S}_{\mathcal{A}}$ of a one-round execution of a protocol $\mathcal{A}$ for three processes in the IFSC model. The complete complex is a disconnected space with an infinite number of connected components. Clearly, this complex is not the same complex of Figure 5.2. Each connected component is associated with an output value of the safe-consensus task. To see why this is true, let us take a closer look at the subcomplex, say $\mathcal{T}_3$, on top of Figure 6.4, it has a 2-simplex that represents a state in the first round of $\mathcal{A}$ that is reachable by means of the round schedule $\mathsf{W}(3), \mathsf{S}(3), \mathsf{R}(3), \mathsf{W}(1,2), \mathsf{S}(1,2), \mathsf{R}(1,2)$. In this execution, process p_3 is faster that p_1 and p_2, thus it executed the safe-consensus object in solo, hence by the Safe-Validity condition of the safe-consensus task, the shared object returns the valid input value of 3 to all processes. Now, if we consider the 2-simplex of $\mathcal{T}_3$ representing the state reachable through the schedule $\mathsf{W}(2,3), \mathsf{S}(2,3), \mathsf{R}(2,3), \mathsf{W}(1), \mathsf{S}(1), \mathsf{R}(1)$ then we know that p_2 and p_3 were faster that p_1 and

[2]In all our pseudocodes, the symbol "*id*" contains the id of the process which is executing the code.

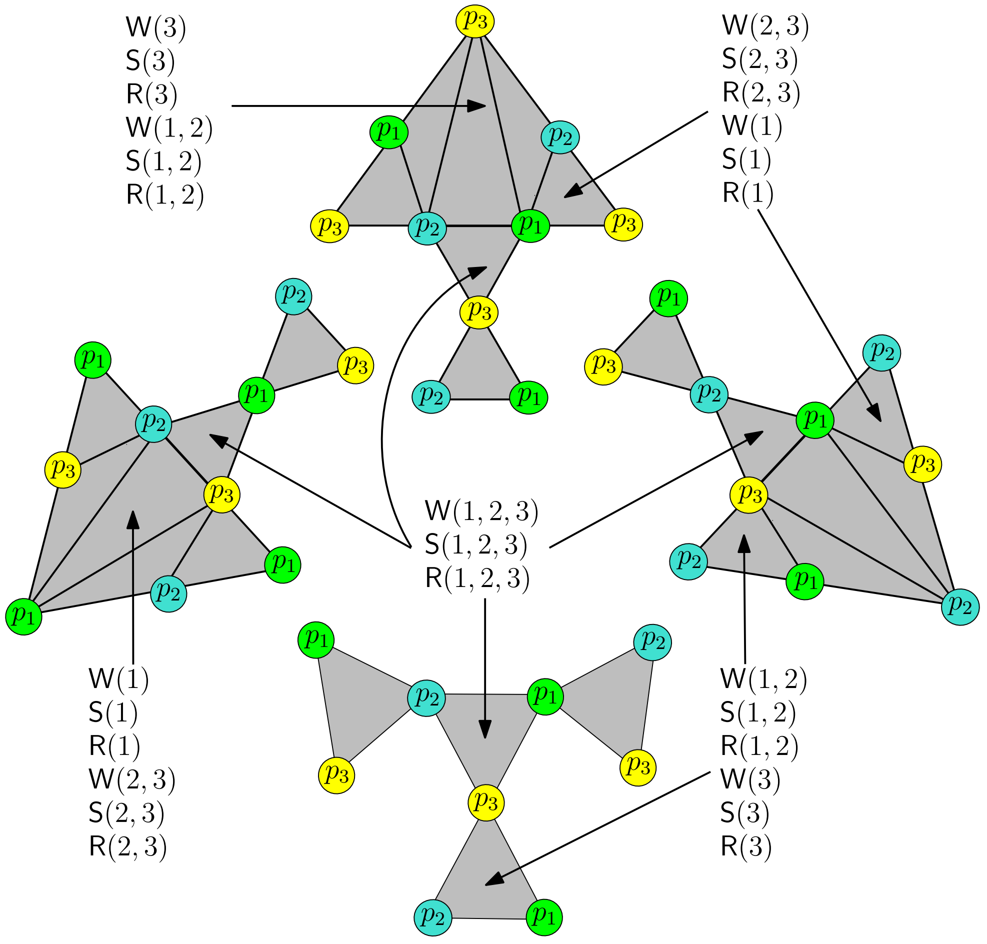


Figure 6.4: Part of the one round Protocol complex of a protocol in the IFSC model. The entire complex contains a countable number of copies of the subcomplex at the bottom.

executed the safe-consensus object concurrently and by the Safe-Validity condition of the safe-consensus task, the shared object can return any value to all processes, so that there exists the possibility that the return value of the safe-consensus object is 3. With a similar analysis of the other simplexes in $\mathcal{T}_3$, we can conclude that the safe-consensus value given in the vertices of every 2-simplex of $\mathcal{T}_3$ is precisely 3. This subcomplex cannot have simplexes with a safe-consensus value other that 3. For example, it is easy to show that the simplex representing the state obtained with the execution of $\mathcal{A}$ given by $\mathsf{W}(1), \mathsf{S}(1), \mathsf{R}(1), \mathsf{W}(2,3), \mathsf{S}(2,3), \mathsf{R}(2,3)$ cannot be adjacent to any simplex of $\mathcal{T}_3$ (because of the different safe-consensus values). If we now take the subcomplex $\mathcal{T}_1$ in the leftmost part of the figure and the subcomplex $\mathcal{T}_2$ in the rightmost part (both of them are isomorphic to $\mathcal{T}_3$), we can prove that $\mathcal{T}_1$ contains only simplexes with vertices of safe-consensus value equal to 1 and all the simplexes of $\mathcal{T}_2$ have vertices with 2 as the safe-consensus value.

What about the subcomplex B_x at the bottom of the figure ? It contains simplexes that represent states in which the safe-consensus object returns an invalid output value $x \neq 1, 2, 3$. All the states represented in B_x come from executions in which at least two processes invoke the safe-consensus task concurrently and the value the processes obtain from the shared object is precisely x. But accordingly to the definition of the safe-consensus task presented in Section 6.1.2, there can be a countable number of invalid output values that the task can return to the processes, so that there should be in the complex of Figure 6.4 a subcomplex B_β (isomorphic to B_x) for each possible invalid value β. For simplicity, we only show one of these subcomplexes. In summary, the protocol complex $\mathcal{S}_\mathcal{A}$ of the first round of the protocol is disconnected, with an infinite number of connected subcomplexes, one for each possible output value of the safe-consensus task.

We have described the one-round protocol complex of a protocol in the IFSC model. Because we work in an iterated model, in the second (third, fourth and so on) round, the protocol complex is composed of many subcomplexes like $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ and B_x, each simplex of the previous round transforms into a subcomplex like $\mathcal{S}_\mathcal{A}$.

It is not hard to prove that the behavior we have described for 3-processes protocols in the IFSC model, generalizes to protocols with any number of processes $n \neq 3$. The protocol complex of any protocol in the IFSC model is a disconnected complex with an infinite number of connected subcomplexes. In general, the structure of the protocol complex in the IFSC model is not as uniform as the well behaved structure of the protocol complex in the basic iterated model. The topology of the protocol complex depends on the way the safe-consensus objects are invoked by the processes (e.g., how many objects are used in the same round and which processes invoke each object). Thus, the structure of the protocol complex of a given round, can be very different from the structure of the previous or next round, even it could happen that there is no relation between these complexes. The case of the IFSC model is one case that allows some simple analysis to be carried on, the general case will require more sophisticated tools for its study.

6.3 Solving set agreement tasks in the IFSC model

We now show that the difference between protocols in the basic iterated model and protocols in the iterated model with safe-consensus is not only about geometric shapes. In this section, we prove that in the IFSC model we can solve the $(n, n-1)$-set agreement task for $n \geqslant 2$ processes, the simplest set agreement task that we know is unsolvable in the iterated model [HS93, SZ93, BG93a, HS99, SZ00]. But we also prove that we cannot solve every set agreement task, because n-consensus ($(n, 1)$-set agreement) is impossible to solve in the IFSC model, when $n \geqslant 3$.

6.3.1 A protocol for $(n, n-1)$-set agreement

The protocol in Figure 6.5 solves $(n, n-1)$-set agreement for $n \geqslant 2$ processes in one round with only one snapshot object and one safe-consensus object. We proceed to prove its correctness.

Lemma 6.3.1. *For any process p_i that executes line 9 of the protocol in Figure 6.5,*

(a) The cardinality of the set $A_i = \{\alpha \mid sm_i[\alpha] \neq \bot\}$ is at least 2 and $\min A_i \neq n$.

(b) If $n \notin A_i$ then $\min A_i \neq n - 1$.

Proof. (a) Suppose that process p_i executes line 9 of the **if/else** block, then it must be true that $val_i \notin \overline{n} \vee sm_i[val_i] = \bot$, so either the safe-consensus object returned an invalid process id or process p_{val_i} did not write its proposed value to the shared memory before p_i executed the snapshot operation (p_{val_i} could be slow or perhaps it crashed). In any case, by (2) of the Safe-Validity condition of the safe-consensus task, there must exists two processes p_r, p_s that called the safe-consensus object concurrently. These processes wrote their input values to the shared memory before accessing the safe-consensus object, so that when p_i takes a snapshot of the memory, the local array sm_i contains at least two non-$\perp$ values and with this we have that $|A_i| \geqslant 2$. Now suppose that $\alpha_i = \min A_i$ is such that $\alpha_i = n$. There must exists $j \in A_i$ with $j \neq n$, but then $j < n = \alpha_i$ which is a contradiction. Hence, $\alpha_i \neq n$. (b) Given that $n \notin A_i$, we can use a similar argument to that used in (a) and obtain this case. □

```
(1)  init $dec, val, sm \leftarrow \perp$;

(2)  begin
(3)      $SM$.update($input$);
(4)      $val \leftarrow safe\text{-}consensus.exec(id)$;
(5)      $sm \leftarrow SM$.scan();
(6)      if $val \in \overline{n}$ and $sm[val] \neq \perp$ then
(7)          $dec \leftarrow sm[val]$;
(8)      else
(9)          $dec \leftarrow sm[\alpha]$ with $\alpha = \min\{\alpha \mid sm[\alpha] \neq \perp\}$;
(10)     end if
(11)     decide $dec$;
(12) end
```

Figure 6.5: A $(n, n-1)$-set agreement protocol in the IFSC model

Theorem 6.3.2. *Let $n \geqslant 2$. The $(n, n-1)$-set agreement task is solvable in the IFSC model in one round using one safe-consensus object.*

Proof. We prove that the protocol in figure 6.5 solves $(n, n-1)$-set agreement. Trivially, the protocol satisfies the Termination condition of the $(n, n-1)$-set agreement task. Let p_i be any process; after p_i writes to the shared memory its input value, it invokes the unique safe-consensus object and takes a snapshot of the memory, p_i executes the **if/else** block at lines 6-10. First suppose that $val_i \in \overline{n}$ and $sm_i[val_i] \neq \perp$. Then process p_{val_i} wrote to the shared memory its input value before p_i took the snapshot and this implies that $sm_i[val_i]$ contains a valid proposed input value and this is the decided value of process p_i. On the other hand, if $sm_i[val_i] = \perp$ or $val_i \notin \overline{n}$, p_i goes on to execute line 9. By (a) of Lemma 6.3.1, the set $A_i = \{\alpha \mid sm_i[\alpha] \neq \perp\}$ is not empty, so that there exists a minimum element $\alpha_i \in A_i$ and then when p_i assigns to dec_i the contents of the local register $sm_i[\alpha_i]$, it has a valid proposed input value, so that the decided value of process p_i is correct. Hence, the Validity condition of the $(n, n-1)$-set agreement task is fulfilled by the protocol.

We now show that the set of values decided by the processes in any execution of the protocol has size no bigger that $n-1$. Suppose that processes $p_{i_1}, \ldots, p_{i_r}$ ($r \leqslant n$, some processes may crash or be delayed) finish an execution of the protocol and let D be the set of values decided by the processes. We argue by cases.

Case 1. Two or more processes executed line 7 of the protocol. Then $|D| \leqslant n-1$, because all processes invoked the same safe-consensus object and by the Safe-Validity property of the safe-consensus task, the return value of the shared object is the same for all processes, so that if at least two processes executed line 7, they decided the same value.

Case 2. Only one process p_l executed line 7 of the protocol. Let v be the value that the safe-consensus object returned to all participating processes. Then for process p_l we have that

$val_l = v$ and $sm_l\,[v]$ (which is non-$\bot$) is the value decided by p_l. If $v \in \{1, \ldots, n-1\}$ then $|D| \leqslant n-1$, because for every process p_i with $i \neq l$, p_i executed the second part of the **if/else** block and by (a) of Lemma 6.3.1, p_i could not decide the value proposed by process p_n. On the other hand, if $v = n$, then for all $i \neq l$, $val_i = n$ and $sm_i\,[val_i] = \bot$ and these two facts imply that $A_i \subseteq \{1, \ldots, n-1\}$. With (b) of Lemma 6.3.1 we have that $\min A_i \neq n-1$, so that all other processes with ids not equal to l decided at most $n-2$ values, which together with the value decided by p_l, make up for at most $n-1$ different values, therefore $|D| \leqslant n-1$.

Case 3. No process executed line 7 of the protocol. By (a) of Lemma 6.3.1, for every process p_i we have that $\alpha_i \in \{1, \ldots, n-1\}$, so all processes decided at most $n-1$ values.

In any case, we conclude that the set of decided values has size at most $n-1$, thus the protocol satisfies the $(n-1)$-Agreement condition of the $(n, n-1)$-set agreement task, hence the protocol is correct. □

6.3.2 Impossibility of consensus in the IFSC model

Theorem 6.3.2 tell us that the IFSC model is more powerful that the basic iterated model, because in the new model we can solve the $(n, n-1)$-set agreement task. But now we will show that in the IFSC model, we cannot solve consensus, i.e., $(n, 1)$-set agreement.

Trying to solve consensus in distributed systems

The impossibility results for consensus in various models of distributed computing [FLP85, LAA87, BMZ90] and the impossibility results for set agreement in the iterated model [HS99, BG93a, SZ00], have shown that solving consensus in distributed environments is related to connectivity of graphs (0-connectivity of simplicial complexes). Roughly speaking, consensus is not solvable in a given model if for each valid input of the consensus task, the protocol complex (associated with that input[3]) is connected.

In Theorem 6.3.6 we will prove that it is not possible to solve the consensus problem for $n \geqslant 3$ processes in the IFSC model, despite the fact that the protocols in this model have disconnected protocol complexes (see Figure 6.4). We remark that this impossibility comes from the restriction in the use of the safe-consensus objects (all processes invoke the same object in each round) of the IFSC model, and not from the fact that we are working in an iterated model. In Chapter 7, we will build a protocol in the iterated model with safe-consensus that implements n-consensus with $\binom{n}{2}$ safe-consensus shared objects.

If $\mathcal{A}$ is a protocol for the consensus task in the IFSC model and its protocol complex $\mathcal{S}_{\mathcal{A}}$ is disconnected: Why is it impossible for $\mathcal{A}$ to solve consensus? We can always find an execution of $\mathcal{A}$ in which when all the processes have decided their output values, there are at least two processes that decided two distinct values, contradicting the Agreement property of the consensus task. Now, this "bad execution" exists because we can choose $i \in \overline{n}$ and two simplexes $\Delta_i, \Delta_{\overline{n}-i}$ in the complex $\mathcal{S}_{\mathcal{A}}$ such that

- Δ_i represents an execution in which process p_i can see only his own input value (that is, an execution in solo of p_i).

[3]Remember our discussion at the end of Section 5.3 about our definition of protocol complexes and its relation with the definition of a span of [HS99].

- $\Delta_{\overline{n}-i}$ represents an execution in which the other processes never see p_i's input value.
- The simplexes Δ_i and $\Delta_{\overline{n}-i}$ lie inside a connected component of $\mathcal{S}_{\mathcal{A}}$.

As a consequence of the last point, there is a (graph) path from any vertex of Δ_i to any vertex in $\Delta_{\overline{n}-i}$. Using one of these paths and an argument involving non-distinguishable states, we can prove that the bad execution exists, so that $\mathcal{A}$ cannot solve consensus. Thus we see that this impossibility is due to some kind of "local connectivity" (between a specific pair of vertices) of $\mathcal{S}_{\mathcal{A}}$.

There is something that the reader will notice immediately in the impossibility proof of consensus in the IFSC model and it is that this proof for the IFSC model is very much the same proof given in Section 5.4.2 about consensus and the impossibility to solve it in the iterated model with shared memory. We will only need to modify one result, namely Lemma 5.4.4 and the used round schedules π_i, π_* and $\pi_{\overline{n}-i}$, all other ingredients will remain exactly the same, as they are valid also for protocols in the IFSC model. Before the formal impossibility proof, we will workout an example with three processes. Let $i \in \overline{n}$, we redefine the round schedules π_i, π_* and $\pi_{\overline{n}-i}$ for the IFSC model in the following way:

(π_i) $\quad \mathsf{W}(i), \mathsf{S}(i), \mathsf{R}(i), \mathsf{W}(\overline{n}-i), \mathsf{S}(\overline{n}-i), \mathsf{R}(\overline{n}-i),$

(π_*) $\quad \mathsf{W}(\overline{n}), \mathsf{S}(\overline{n}), \mathsf{R}(\overline{n}),$

$(\pi_{\overline{n}-i})$ $\mathsf{W}(\overline{n}-i), \mathsf{S}(\overline{n}-i), \mathsf{R}(\overline{n}-i), \mathsf{W}(i), \mathsf{S}(i), \mathsf{R}(i).$

These round schedules are almost the same as their counterparts in the proof of Theorem 5.4.7 of Chapter 5, the only difference being that we have added the call to the unique safe-consensus shared object between the write and read shared memory operations.

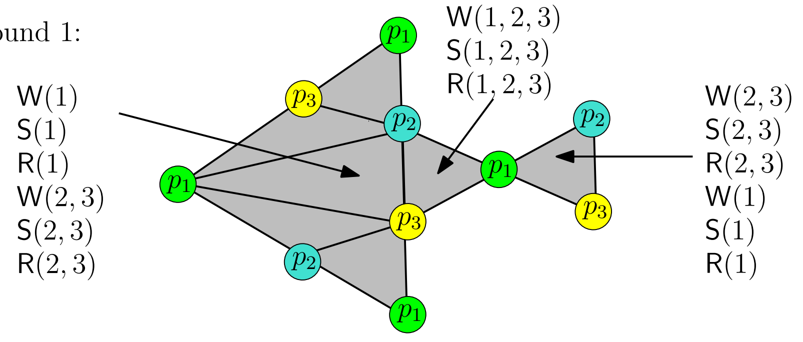
An example with three processes

If we take the case of $n = 2$ processes, then by Theorem 6.3.2, consensus of 2 processes ((2, 1)-set agreement task) is solvable with safe-consensus objects. As we have seen in this section, the protocol complexes of protocols in the IFSC model are disconnected, so that one would expect to be able to solve consensus in this model, but our next results argue otherwise.

Let $\mathcal{A}$ be a protocol for three processes in the IFSC model and assume that $\mathcal{A}$ can solve the consensus task and for simplicity, suppose that the processes receive as input values their own ids. Let I be the initial state in which the processes begin to execute the protocol. If the processes run $\mathcal{A}$ only one round and then decide a (unique) output value, then we know that the protocol complex $\mathcal{S}_{\mathcal{A}}$ is the space shown in Figure 6.4. One of the subcomplexes of $\mathcal{S}_{\mathcal{A}}$ is depicted at the top of Figure 6.6, it represents all the reachable states in the first round of $\mathcal{A}$ for which the output value of the safe-consensus object is 1. The round schedules π_1, π_* and $\pi_{\overline{3}-1}$ give three possible ways to execute the first round of $\mathcal{A}$ and they take the protocol into the states $I \cdot \pi_1, I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$ respectively and these states are represented in the complex of Figure 6.6. Because the value returned by the safe-consensus object to all processes in the three states is 1, it is easy to see that p_2 and p_3 cannot distinguish between $I \cdot \pi_1$ and $I \cdot \pi_*$ and p_1 cannot distinguish between $I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$, thus we have the path of connected states

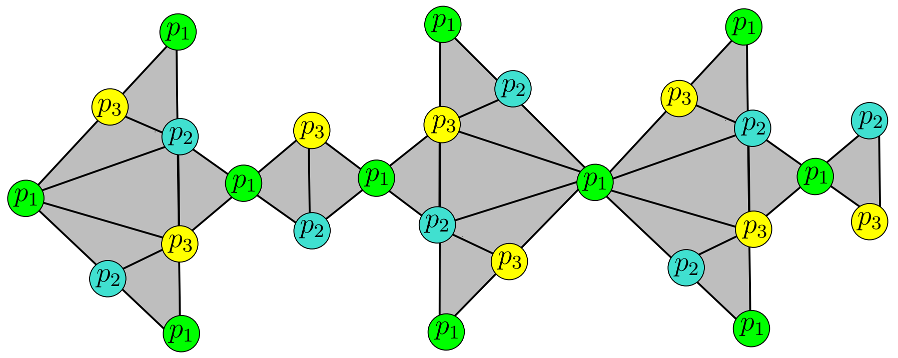
$$I \cdot \pi_1 \overset{\{2,3\}}{\sim} I \cdot \pi_* \overset{1}{\sim} I \cdot \pi_{\overline{3}-1}.$$

Round 1:



$$I \cdot \pi_1 \overset{\{2,3\}}{\sim} I \cdot \pi_* \overset{1}{\sim} I \cdot \pi_{\overline{3}-1}$$

Round 2:



$$(I \cdot \pi_1) \cdot \pi_1 \overset{\{2,3\}}{\sim} (I \cdot \pi_1) \cdot \pi_* \sim \cdots \sim (I \cdot \pi_{\overline{3}-1}) \cdot \pi_* \overset{1}{\sim} (I \cdot \pi_{\overline{3}-1}) \cdot \pi_{\overline{3}-1}$$

Figure 6.6: Subcomplexes of the first and second round protocol complexes of a protocol in the IFSC model.

If the processes solve consensus in just one round, then these states are decision states. As in the state $I \cdot \pi_1$ process p_1 can see only itself, it has to decide its own input value, which is 1 and by the Agreement property of consensus, p_2 and p_3 are forced to decide 1; while in the state $I \cdot \pi_{\overline{3}-1}$, p_2 and p_3 cannot see the input value of p_1 and this implies that p_2, p_3 decide a unique element u of the set $\{2, 3\}$ and p_1 also decides u. As we have said before, p_2 and p_3 cannot distinguish between the states $I \cdot \pi_1$ and $I \cdot \pi_*$ and they decide 1 in the state $I \cdot \pi_1$, then they also decide 1 in $I \cdot \pi_*$. On the other hand, p_1 cannot distinguish between $I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$ and in the later state p_1 decides u, thus p_1 also decides u in $I \cdot \pi_*$. We conclude that in the execution in which $\mathcal{A}$ enters the state $I \cdot \pi_*$, p_2 and p_3 decide 1 but p_1 decides $u \neq 1$. This constitutes a violation of the Agreement condition of the consensus task, then the protocol cannot end in one iteration. If the processes execute $\mathcal{A}$ for one more round and decide, then because we work in an iterated model, we can prove that $\mathcal{S}_{\mathcal{A}}$ will contain the subcomplex given at the bottom of Figure 6.6; it has simplexes representing reachable states in the second round of $\mathcal{A}$ of the form

$$(I \cdot \rho) \cdot \sigma \qquad \rho, \sigma \in \{\pi_1, \pi_*, \pi_{\overline{3}-1}\},$$

and it can be verified that this complex contains a path of states that connect $(I \cdot \pi_1) \cdot \pi_1$ with

$(I \cdot \pi_{\overline{3}-1}) \cdot \pi_{\overline{3}-1}$ and this fact can be used to show that there is an execution of the second round of $\mathcal{A}$ in which processes decide two different values, again violating the Agreement property of consensus. As we work in an iterated model, we can repeat the same argument in every round of $\mathcal{A}$ and that would imply that $\mathcal{A}$ cannot solve the consensus task for three processes.

The formal results

We now formalize the main ideas of the example above to prove that it is impossible to solve n-consensus ($n \geqslant 3$) in the IFSC model. Our first result is the IFSC-version of Lemma 5.4.4

Lemma 6.3.3. *Let $n \geqslant 3, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the IFSC model and S is any reachable state in some round $r \geqslant 0$ of $\mathcal{A}$, then there exists the path*

$$S \cdot \pi_i \overset{\overline{n}-i}{\sim} S \cdot \pi_* \overset{i}{\sim} S \cdot \pi_{\overline{n}-i}, \tag{6.1}$$

of connected reachable states in round $r+1$ of $\mathcal{A}$. Moreover, the value of the safe-consensus object is the same in all three states.

Proof. Let S be a reachable state in $\mathcal{A}$ and $i \in \overline{n}$. We first show that there exists executions of $\mathcal{A}$ in which the return value of the safe-consensus object is the same in the three states $S \cdot \pi_i, S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$ respectively. If this fact is true, then it will be clear that the given states can be connected in the way described by (6.1).

In the state $S \cdot \pi_i$, the value of the safe-consensus object is i, because p_i executes the shared object before any other process executes it and by the Validity condition of the safe-consensus task, the return value must be the value proposed by p_i. For the states $S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$, the value of the safe-consensus can be arbitrary. This is true because in each round, all processes invoke the same safe-consensus object and by hypothesis $n \geqslant 3$ (this implies that $|\overline{n}|, |\overline{n} - i| \geqslant 2$). Thus in the executions of $\mathcal{A}$ (given by the round schedules π_* and $\pi_{\overline{n}-i}$) that take the protocol into the states $S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$, at least two processes execute the safe-consensus object concurrently. By the Validity property, the return value of the object can be arbitrary. Therefore there exists the possibility that the value returned by the safe-consensus object to all processes is precisely i in the states $S \cdot \pi_i, S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$. It follows that the path (6.1) of connected reachable states in $\mathcal{A}$ exists. □

Lemma 6.3.4. *Let $n \geqslant 3, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the IFSC model and there exists a path*

$$S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l \qquad (l \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that for all j with $1 \leqslant j \leqslant l$, $X_j = \overline{n} - i$ or $X_j = \{i\}$. Then there exists a path

$$Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_s}{\sim} Q_s \qquad (s \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that

(1) Every state Q_t is a successor state of some S_j.

(2) For all m with $1 \leqslant m \leqslant s$, Z_m is such that $Z_m = \overline{n} - i$ or $Z_m = \{i\}$.

Proof. This proof is identical to the proof of Lemma 5.4.5, we only need to replace the use of Lemma 5.4.5 with Lemma 6.3.4. □

Lemma 6.3.5. *Let $n \geqslant 3, i \in \overline{n}$. Suppose that $\mathcal{A}$ is a protocol in the IFSC model that solves the consensus task and that $S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l$ $(l \geqslant 1)$ is a path of connected reachable states in $\mathcal{A}$ that satisfies the hypothesis of Lemma 6.3.4 and also assume that S_0 is a v-valent state. Then S_l is v-valent.*

Proof. Again, the proof is the same as that of Lemma 5.4.6, replacing any reference to Lemmas 5.4.4 and 5.4.5 by Lemmas 6.3.3 and 6.3.4. □

Theorem 6.3.6. *For $n \geqslant 3$, there is no protocol in the IFSC model to solve consensus for n processes.*

Proof. Replace Lemma 5.4.6 with Lemma 6.3.5 in the proof of Theorem 5.4.7 to obtain this one and we are done. □

The results that we have obtained in Theorems 6.3.2 and 6.3.6 tell us two important facts: (1) that the IFSC model is indeed more powerful that the basic iterated model, because we can solve the $(n, n-1)$-set agreement task, which is the simplest set agreement instance that is known to be impossible to solve in the iterated model [BG93a, HS93, SZ93]; (2) The IFSC model is not that powerful, because although $(n, n-1)$-set agreement is solvable, the consensus task cannot be implemented in the IFSC model. Thus, the IFSC model is still limited in computational power, for task solvability. In the next chapter, we will prove that the limitations of the IFSC model comes from the restriction on the way the processes invoke the only safe-consensus object available in each iteration. We will show that in the general extended iterated model with safe-consensus objects, the consensus task can be implemented with safe-consensus, with an iterated protocol with very nice combinatorial features.

An interesting open question concerning the IFSC model is the following: Is the $(n, n-2)$-set agreement task solvable in the IFSC model ? We conjecture that the answer is no. We believe that topological methods (different from tools like Sperner's Lemma) will be necessary to give an answer to this question.

7 Implementing consensus in the iterated model with safe-consensus

In this chapter, we implement the consensus task using safe-consensus objects, with a protocol in the iterated model with safe-consensus. Although the formal specification of this protocol is a bit complicated (see Section 7.2.2), the intuitive idea behind it is quite simple and this idea can be depicted in a graphical way. This is a nice property of our consensus protocol and it is a consequence of working in an (extended) iterated model of computation. As a byproduct of the work in this chapter, we define a new distributed task, which is a new kind of "consensus task". More details can be found in Section 7.1.

From this chapter to the end of this thesis, we focus our attention entirely on the consensus task and the iterated model with safe-consensus objects.

7.1 Coalitions consensus tasks

In order to give the complete specification of the consensus iterated protocol presented in Section 7.2.2, we first introduce an "intermediate" task, the *g-2coalitions-consensus* task, in it, g processes try to decide a consensus value from two proposed input values, but there are exactly $g-2$ processes which know the two input values, while each of the two remaining processes know only one input value. As a first result towards proving the correctness of the consensus protocol of Section 7.2.2, we show that the g-2coalitions-consensus task can be implemented with a one round iterated protocol using only one safe-consensus shared object.

7.1.1 Definition of the g-2coalitions-consensus task

To introduce formally the 2coalitions-consensus task, we need to define the set of valid input values for this new task. Let I be a non-empty set and $\mathcal{N} = I \cup \{\perp\}$ ($\perp \notin I$). If $x = \langle v_1, v_2 \rangle \in \mathcal{N} \times \mathcal{N}$, $x.left$ denotes the value v_1 and $x.right$ is the value v_2.

Definition 7.1.1. An ordered g-tuple $C = (x_1, \ldots, x_g) \in (\mathcal{N} \times \mathcal{N})^g$ is a *g-coalitions tuple* if and only if the following conditions are fulfilled.

(a) For all x_i in C, $x_i.left \neq \perp$ or $x_i.right \neq \perp$.

(b) If x_i, x_j are members of C such that $x_i.left \neq \perp$ and $x_j.left \neq \perp$, then $x_i.left = x_j.left$. A similar rule must hold if $x_i.right \neq \perp$ and $x_j.right \neq \perp$.

Let $l(C) = \{i \in \overline{g} \mid x_i.left \neq \perp\}$ and $r(C) = \{j \in \overline{g} \mid x_j.right \neq \perp\}$. Then we also require that

(c) $|l(C) - r(C)| = 1$ and $r(C) - l(C) = \{g\}$.

A more general task can be defined if we omit[1] property (c), but for our purposes, this definition of the g-2coalitions-consensus task is enough. Notice that property (a) implies that $l(C) \cup r(C) = \overline{g}$. The set of all g-coalitions tuples is denoted by $\mathcal{C}_g$. Some examples can help us understand better the concept of a coalition tuple. Let $\mathcal{N} = \{0, 1, \perp\}$. The following tuples are examples of elements of the set $\mathcal{C}_4$,

$$C_1 = (\langle 1, \perp\rangle, \langle 1, 0\rangle, \langle 1, 0\rangle, \langle \perp, 0\rangle);$$

$$C_2 = (\langle 1, 0\rangle, \langle 1, 0\rangle, \langle 1, \perp\rangle, \langle \perp, 0\rangle).$$

Clearly, $C_1, C_2 \in \mathcal{C}_4$ and also we have that $l(C_1) = \{1, 2, 3\}$ and $r(C_1) = \{2, 3, 4\}$. Notice that $l(C_1) \cup r(C_1) = \overline{4}$ and also $l(C_1) \cap r(C_1) = \{2, 3\}$. For C_2, it is true that $l(C_2) - r(C_2) = \{1, 2, 3\} - \{1, 2, 4\} = \{3\}$, $r(C_2) - l(C_2) = \{1, 2, 4\} - \{1, 2, 3\} = \{4\}$ and $l(C_2) \cap r(C_2) = \{1, 2\}$. The following tuples are not elements of any coalitions tuples set:

$$C_3 = (\langle \perp, \perp\rangle, \langle 1, 0\rangle, \langle 1, 0\rangle, \langle 1, 0\rangle);$$

$$C_4 = (\langle 1, \perp\rangle, \langle 1, 0\rangle, \langle \perp, 1\rangle, \langle \perp, 0\rangle).$$

For C_3, properties (a) and (c) of Definition 7.1.1 are not satisfied and for C_4, condition (b) is not fulfilled. Thus we conclude that $C_3, C_4 \notin \mathcal{C}_4$. We can now define a new distributed task.

The g-2coalitions-consensus task. We have g processes $p_1, \ldots, p_g$ and each process p_i starts with a private input value of the form $x_i \in \mathcal{N} \times \mathcal{N}$ such that $C = (x_1, \ldots, x_g) \in \mathcal{C}_g$, and C is called a *global input*. In any execution, Termination and Agreement must be satisfied, and in addition:

- g-2coalitions-Validity: If some process outputs c, then there must exists a process p_j with input x_j such that $x_j = \langle c, v\rangle$ or $x_j = \langle v, c\rangle$ ($c \in I, v \in \mathcal{N}$).

Notice that the sets $\mathcal{C}_g$ ($g \geqslant 2$) encapsulate the intuitive ideas given at the beginning of this chapter about the coalition-consensus task that we need: With an input $C \in \mathcal{C}_g$, there are two processes that known exactly one of the input values, while the rest of the processes do known the two input values. These processes could easily reach a consensus, but they still need the other two processes with unique input values to agree on the same value, and this is the difficult part of the g-2coalitions-consensus task.

7.1.2 Safe-consensus implements g-2coalitions-consensus

We now need to justify that we can implement the task g-2coalitions-consensus with a single-round protocol using one snapshot object and one safe-consensus object. For this purpose, we have the protocol given in Figure 7.1. Each process p_i receives as input a tuple with values satisfying the properties of the 2coalitions-consensus task and then in lines 3-5, p_i writes its input tuple in shared memory using the snapshot object SM; invokes the safe-consensus object with its id as input, storing the unique output value u of the shared object in the local variable *val* and finally, p_i takes a snapshot of the memory. Later, what happens in Lines 6-10 depends on the output value

[1]The condition (c) in the definition of coalitions tuple is needed because, without it, safe-consensus cannot implement g-2coalitions-consensus with one safe-consensus object. It is not hard to prove this fact, so that we omit it.

u of the safe-consensus object. If $u = g$, then by the Safe-Validity property, either p_g invoked the object or at least two processes invoked the safe-consensus object concurrently and as there is only one process with input tuple $\langle v, \perp \rangle$, p_i will find an index j with $sm[j].right \neq \perp$ in line 7, assign this value to dec and in line 11 p_i decides. On the other hand, if $u \neq g$, then again by the Safe-Validity condition of the safe-consensus task, either process p_u is running and invoked the safe-consensus object or two or more processes invoked concurrently the shared object and because all processes with id not equal to g have input tuple $\langle z, y \rangle$ with $z \neq \perp$, it is guaranteed that p_i can find an index j with $sm[j].left \neq \perp$ and assign this value to dec to finally execute line 11 to decide its output value. All processes decide the same value because of the properties of the input tuples of the 2coalitions-consensus task and the Agreement property of the safe-consensus task.

```
(1)  procedure $g$-2coalitions-consensus($v_1, v_2$)
(2)  begin
(3)      $SM$.update($\langle v_1, v_2 \rangle$);
(4)      $val \leftarrow safe\text{-}consensus.exec(id)$;
(5)      $sm \leftarrow SM$.scan();
(6)      if $val = g$ then
(7)          $dec \leftarrow$ choose any $sm[j].right \neq \perp$;
(8)      else
(9)          $dec \leftarrow$ choose any $sm[j].left \neq \perp$;
(10)     end if
(11)     decide $dec$;
(12) end
```

Figure 7.1: A g-2coalitions-consensus protocol with one safe-consensus object.

Lemma 7.1.2. *The protocol of Figure 7.1 solves the g-2coalitions-consensus task using one snapshot object and one safe-consensus object.*

Proof. Let $p_i \in \{p_1, \ldots, p_g\}$; after p_i writes the tuple $\langle v_1, v_2 \rangle$ to the snapshot object SM, it invokes the safe-consensus object and takes a snapshot of the shared memory, p_i enters into the **if**/**else** block at lines 6-10. Suppose that the test in line 6 is successful, this says that the safe-consensus object returned the value g to process p_i. By the Safe-Validity condition of the safe-consensus task, either process p_g invoked the shared object or at least two processes p_j, p_k invoked the safe-consensus object concurrently (and it could happen that $j, k \neq g$). In any case, the participating processes wrote their input tuples to the snapshot object SM before accessing the safe-consensus object, so that p_i can see these values in its local variable sm_i, and remember that the coalitions tuple C consisting of all the input values of processes $p_1, \ldots, p_g$ satisfies the properties

$$|l(C) - r(C)| = 1 \text{ and } r(C) - l(C) = \{g\}.$$

Thus, the equation $|l(C) - r(C)| = 1$ tell us that only one process has input tuple $\langle x, \perp \rangle$ $(x \neq \perp)$, then when p_i executes line 7, it will find a local register in sm_i such that $sm_i[j].right \neq \perp$ and this value is assigned to dec_i. Finally, p_i executes line 11, where it decides the value stored in dec_i and this variable contains a valid input value proposed by some process.

If the test at line 6 fails, the argument to show that dec_i contains a valid proposed input value is very similar to the previous one. This proves that the protocol satisfies the 2coalitions-Validity condition of the g-2coalitions-consensus task.

The Agreement condition is satisfied because by the Agreement condition of the safe-consensus task, all participating processes receive the same output value from the shared object and therefore all processes have the same value in the local variables val_i $(1 \leqslant i \leqslant g)$, which means that all participating processes execute line 7 or all of them execute line 9. Then, for every process p_r, dec_r contains the value $sm_r\left[j_r\right].right$ or the value $sm_r\left[j'_r\right].left$ where j_r (j'_r) depend on p_r. But because the input pairs of the processes constitute a coalitions tuple C, they satisfy property (b) of Definition 7.1.1 of coalitions tuple, which implies that $sm_r\left[j_r\right].right = sm_r\left[j_q\right].right$ whenever $sm_r\left[j_r\right].right$ and $sm_r\left[j_q\right].right$ are non-$\perp$ (a similar statement holds for the left sides). We conclude that all processes assign to the variables dec_i $(1 \leqslant i \leqslant g)$ the same value and thus all of them decide the same output value, that is, the Agreement property of the g-2coalitions-consensus tasks is fulfilled. The Termination property is clearly satisfied. □

7.2 An iterated consensus protocol

In this section, we build an iterated protocol that solves the consensus task using safe-consensus objects. The full specification of such protocol is in Figure 7.3, but before trying to understand the pseudocode of the protocol, it is a good idea to describe which are the intuitive actions which allow the protocol to solve consensus with the safe-consensus task. We first give an informal short description of how the protocol works and after that, we present the full pseudocode of the protocol in Figure 7.3 (using the g-2coalitions-consensus task).

7.2.1 Basic properties of the consensus protocol

A simple way to describe the protocol that solves consensus is by seeing it as a protocol in which the processes use a set of $\binom{n}{2}$ shared objects, where each of these objects represent a g-2coalitions-consensus task, which by Lemma 7.1.2, can be implemented using one snapshot object and one safe-consensus object.

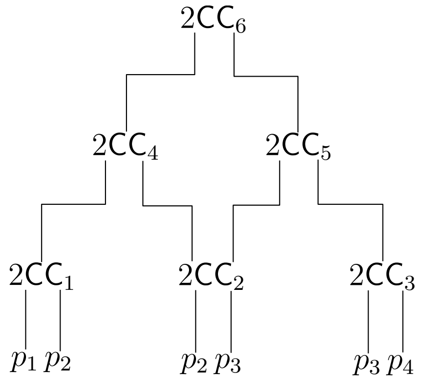


Figure 7.2: The structure of the 4-consensus protocol using 2coalitions-consensus tasks.

Using the task g-2coalitions-consensus, the protocol in Figure 7.3 can be described graphically as shown in Figure 7.2, for the case of $n = 4$. In each round of the protocol, some processes invoke a 2coalitions-consensus object, represented by the symbol 2CC$_i$. In round one, p_1 and p_2 invoke the object 2CC$_1$ with input values $\langle v_1, \perp\rangle$ and $\langle \perp, v_2\rangle$ respectively, (where v_i is the initial input value of process p_i) and the consensus output u_1 of 2CC$_1$ is stored by p_1 and p_2 in some local variables.

In round two, p_2 and p_3 invoke the $2\mathrm{CC}_2$ object with inputs $\langle v_2, \perp\rangle$ and $\langle \perp, v_3\rangle$ respectively and they keep the output value u_2 in local variables. Round three is executed by p_3 and p_4 in a similar way, to obtain the consensus value u_3 from the 2coalition-consensus object $2\mathrm{CC}_3$. At the beginning of round four, p_1, p_2 and p_3 gather the values u_1, u_2 obtained from the objects $2\mathrm{CC}_1$ and $2\mathrm{CC}_2$ to invoke the $2\mathrm{CC}_4$ 2coalition-consensus object with the input values $\langle u_1, \perp\rangle, \langle u_1, u_2\rangle$ and $\langle \perp, u_2\rangle$ respectively (Notice that p_2 uses a tuple with both values u_1 and u_2) and they obtain a consensus value u_4. Similar actions are taken by the processes p_2, p_3 and p_4 in round five with the shared object $2\mathrm{CC}_5$ and the values u_2, u_3 to compute an unique value u_5. Finally, in round six, all processes invoke the last shared object $2\mathrm{CC}_6$, with the respective input tuples

$$\langle u_4, \perp\rangle, \langle u_4, u_5\rangle, \langle u_4, u_5\rangle, \langle \perp, u_5\rangle,$$

and the shared object returns to all processes a unique output value u, which is the decided output value of all processes, thus this is the final consensus of the processes. The case of 4 processes easily generalizes to any value of n.

7.2.2 The protocol

The formal spec of the iterated consensus protocol with safe-consensus objects is given in Figure 7.3. This is a protocol that implements consensus using only $\binom{n}{2}$ 2coalitions-consensus tasks. If we suppose that the protocol is correct, then we can use the g-2coalitions-consensus protocol presented in Section 7.1 (Figure 7.1) to replace the call to the 2coalitions-consensus objects in Figure 7.3 to obtain a full iterated protocol that solves the consensus task using $\binom{n}{2}$ safe-consensus objects. We use the 2coalitions-consensus tasks in the protocol only to make it simpler and easier to understand.

We now give a short description of how the protocol works. There are precisely $\binom{n}{2}$ rounds executed by the processes and in each round, some subset of processes try to agree in a new consensus value among two given input values. The local variables $step$, $firstid$ and $lastid$ are used by the processes to store information that tell them which is the current set of processes that must try to reach a new agreement in the current round, using a 2coalitions-consensus object. The local array *agreements* contains enough local registers used by the processes to store the agreements made in each round of the protocol and two distinct processes can have different agreements stored in the registers of the array *agreements*. Each consensus value v stored in a register of *agreements* is associated with two integers $i_1, i_r \in \overline{n}$ ($r \geqslant 1$), which represent a set of processes $p_{i_1}, \ldots, p_{i_r}$ (with $i_1 < \cdots < i_r$) that have invoked a 2coalitions-consensus object to agree on the value v, thus we can say that v is an agreement made by the coalition of processes represented by the pair (i_1, i_r). To be able to store v in the array *agreements*, the processes use a deterministic function $\mathbf{tup}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which maps bijectively $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$. This map can be easily constructed, for example, here is a simple definition for **tup**

$$\mathbf{tup}(i, j) = \binom{i + j + 1}{2} + j. \qquad (7.1)$$

Using all the elements described above, the processes can use the protocol of Figure 7.3 to implement consensus using $\binom{n}{2}$ 2coalitions-consensus objects, building in the process the structure depicted in Figure 7.2. From the first round and up to round $n - 1$, all the processes use their proposed input values to make new agreements in pairs, in round one, p_1, p_2 invoke a 2coalitions-consensus shared object to agree on a common value, based on their input values; in round 2, p_2

```
(1)  init $step, firstid, lastid \leftarrow 1$;
         $C, D, dec, newagreement \leftarrow \perp$;
         $agreements\,[\mathbf{tup}(id, id)] \leftarrow input$;

(2)  begin
(3)      for $r \leftarrow 1$ to $\binom{n}{2}$
(4)          $lastid \leftarrow firstid + step$;
(5)          if $firstid \leqslant id \leqslant lastid$ then
(6)              $C \leftarrow agreements\,[\mathbf{tup}(firstid, lastid - 1)]$;
(7)              $D \leftarrow agreements\,[\mathbf{tup}(firstid + 1, lastid)]$;
(8)              $newagreement \leftarrow 2coalitions\text{-}consensus\,[r]\,(C, D)$;
(9)              $agreements\,[\mathbf{tup}(firstid, lastid)] = newagreement$;
(10)         end if
(11)         if $lastid < n$ then
(12)             $firstid = firstid + 1$;
(13)         else if $firstid > 1$ then
(14)             $firstid = 1$;
(15)             $step = step + 1$;
(16)         end if
(17)     end for
(18)     $dec \leftarrow agreements\,[\mathbf{tup}(1, n)]$;
(19)     decide $dec$;
(20) end
```

Figure 7.3: An iterated consensus protocol using g-2coalitions-consensus objects.

and p_3 do the same with another 2coalitions-consensus object and their own input values; later, the turn comes to p_3 and p_4 to do the same with another shared object and their input values and so on until the round number $n-1$, where p_{n-1} and p_n agree on a common value in the way that we have already described. All the agreements obtained in these $n-1$ rounds are stored on the local registers

$$\begin{gathered} agreements_1\,[\mathbf{tup}(1,2)]\,, agreements_2\,[\mathbf{tup}(1,2)]\,, agreements_2\,[\mathbf{tup}(2,3)]\,, \\ agreements_3\,[\mathbf{tup}(2,3)]\,, agreements_3\,[\mathbf{tup}(3,4)]\,, agreements_4\,[\mathbf{tup}(3,4)]\,, \\ \dots, \\ agreements_{n-1}\,[\mathbf{tup}(n-1,n)]\,, agreements_n\,[\mathbf{tup}(n-1,n)]\,. \end{gathered} \tag{7.2}$$

In this way, the processes build the first part of the structure shown in Figure 7.2. At the end of round $n-1$, in lines 14-15, each process updates the values of the local variables $step$ and $firstid$ and proceeds to round n. What happens in the next $n-2$ rounds, is very similar to the case of the previous $n-1$ rounds, but instead of making agreements in pairs, the processes reach new agreements in groups of three processes (see Figure 7.2) invoking new 2coalitions-consensus shared objects and the consensus values obtained in the first $n-1$ rounds and when a process reaches round $n-1+n-2 = 2n-3$, it updates the values of its local variables $step$ and $firstid$ and then that process proceeds to round $2n-2$.

In general, when the processes are about to begin executing round $(\sum_{j=1}^{m} n-j)+1$ ($m < n$), they will try to make $n-(m+1)$ new agreements in groups of size $m+2$, with the aid of the

2coalitions-consensus objects and the agreements they obtained from the previous $n - m$ rounds and store the new consensus values in their local arrays *agreements*, using the **tup** function. When a process finishes round $(\sum_{j=1}^{m+1} n - j)$, the values of *step* and *firstid* are updated to repeat the described behavior for the next $n - (m + 2)$ rounds, until the $\binom{n}{2}$ round, where the last agreement is made and this value is the output value of all the processes.

7.3 Correctness of the consensus protocol

We are ready to prove the correctness of our iterated protocol for consensus. The first step is to introduce some technical definitions.

Definition 7.3.1. With respect to the protocol of Figure 7.3, let $F = \{i_1, \ldots, i_r\} \subseteq \overline{n}$ be elements such that $i_1 < \cdots < i_r$. The set F will be called a *coalition* and will be denoted by $F = [i_1, \ldots, i_r]$. Let p_i be a process such that $i \in F$. We say that *p_i belongs to the coalition F* if and only if the following condition is fulfilled:

(1) $agreements_i\,[\mathbf{tup}(i_1, i_r)] = v$, where $v \neq \perp$ is a valid input value proposed by some participating process.

We ask the following *agreement* property to be satisfied by F:

(2) if p_i, p_j are processes which belong to F and v, v' are the values which satisfy (1) for p_i and p_j respectively, then $v = v'$.

The value v of the first condition is called the *name* of the coalition. The second condition says that all processes that belong to the same coalition must agree on the coalition's name. For $n > m \geqslant 0$, let $\gamma(n, m)$ be defined by

$$\gamma(n,m) = \begin{cases} 0 & \text{if } m = 0, \\ \gamma(n, m-1) + n - m & \text{if } m > 0. \end{cases}$$

Notice that $\gamma(n, m) = \sum_{i=1}^{m} n - i$ for $m > 0$ and $\gamma(n, n-1) = \binom{n}{2}$.

7.3.1 Invariants of the protocol

Before proving that the protocol in Figure 7.3 implements consensus, We prove some useful invariants of the protocol that will be most helpful.

Lemma 7.3.2. *Let $1 \leqslant m \leqslant n - 1$ and $1 \leqslant c \leqslant n - m$. If p_i is executing the protocol at the beginning of round number $r = \gamma(n, m-1) + c$ (before line 11) then $step_i = m$ and $firstid_i = c$.*

Proof. We prove this by induction on m. An easy analysis of the code in Figure 7.3 shows that the base case holds (when $m = 1$, in the first $n - 1$ rounds). Assume that for $m < n - 1$, the lemma is true. We first prove the following claim: When p_i starts executing round $\gamma(n, m) + 1$, $step_i = m + 1$ and $firstid_i = 1$. *Proof of the claim*: By the induction hypothesis, when process p_i executed the protocol at round $r' = \gamma(n, m) = \gamma(n, m-1) + c = \gamma(n, m-1) + (n - m)$ before line 11, the local variables $step_i$ and $firstid_i$ had the values of m and $n - m$ respectively. When p_i reached line 11, it executed the test of the **if** statement, but before that, p_i executed line 4 of the protocol, so that $lastid_i = firstid_i + step_i = (n-m) + m = n$; thus the test in line 11 failed and p_i executed lines 14-15 of the **else** statement ($firstid_i > 1$ because $m < n-1$) and then $step_i$ was incremented by one and $firstid_i$ was set to 1 at the end of round r'. Therefore, when p_i starts executing round $\gamma(n,m)+1$, $step_i = m+1$ and $firstid_i = 1$.

Now suppose that p_i executes the protocol at the beginning of round number $r = \gamma(n,m) + c$ where $1 \leqslant c \leqslant n - (m+1)$. If $c = 1$ then by the preceding argument, $step_i = m+1$ and $firstid_i = 1 = c$. Using this as a basis for an inductive argument, we can prove that for $c \in \{1, \dots, n-(m+1)\}$, $firstid_i = c$ and $step_i = m+1$. □

Lemma 7.3.3. *Let $0 \leqslant m \leqslant n-2$. Suppose that process p_i is about to begin executing the protocol at round $r = \gamma(n,m) + c$ $(1 \leqslant c \leqslant n-(m+1))$. If $c \leqslant i \leqslant c+m+1$ and p_i belongs to the coalition $P = [c, \dots, c+m]$ or it belongs to the coalition $Q = [c+1, \dots, c+m+1]$, then at the end of round r, p_i belongs to the coalition $[c, \dots, c+m+1]$.*

Proof. Let p_i be any process that begins executing round r. By Lemma 7.3.2, we know that $step_i = m+1$ and $firstid_i = c$, which implies that $lastid_i = c+m+1$. If $i \in \{c, \dots, c+m+1\}$, the **if**'s test at line 5 is successful and after that what happens in lines 6,7 depends on i. If $c \leqslant i \leqslant c+m$, then p_i is in the coalition P and if $c+1 \leqslant i \leqslant c+m+1$, p_i is in the coalition Q, thus a valid input value is assigned to at least one of the variables C_i or D_i, so p_i invokes in line 8 the $(m+2)$-2coalitions-consensus object with the input $x_i = \langle C_i, D_i \rangle$. We now pause for a moment to argue that the tuple

$$J = (x_c, \dots, x_{c+m+1})$$

composed of the inputs of processes $p_c, \dots, p_{c+m+1}$ is a valid global input for the $(m+2)$-2coalitions-consensus task. Indeed, from the hypothesis, it is easy to see that J satisfies the requirements (a)-(c) of Definition 7.1.1 and notice that $r(J) - l(J) = \{c+m+1\}$ and $l(J) - r(J) = \{c\}$.

Now back to the execution of process p_i. After p_i invokes the $(m+2)$-2coalitions-consensus task, the output value of the shared object, say v, is assigned to the local variable *newagreement$_i$*. Finally, p_i stores the contents of *newagreement$_i$* in the local array *agreements$_i$* at position $\mathbf{tup}(firstid_i, lastid_i)$, where $firstid_i = c$ and $lastid_i = c+m+1$. Because of the Agreement condition of the 2coalitions-consensus task, every process with id in the set $\{c, \dots, c+m+1\}$ obtained the same value v as return value from the shared object and by the 2coalitions-Validity, this is a valid input value proposed by some process. Therefore properties (1) and (2) of Definition 7.3.1 are satisfied, thus $R = [c, \dots, c+m+1]$ is a valid coalition and process p_i belongs to the coalition R at the end of round r.

On the other hand, if $i \notin \{c, \dots, c+m+1\}$, p_i does not executes the body of the **if** statement (lines 6 to 9) and it goes on to execute the **if**/**else** block at lines 11-16 and then round r ends for p_i, thus it does not try to make a new coalition with other processes. □

Lemma 7.3.4. *Let $m \in \{0, \dots, n-2\}$ be fixed. Suppose that process p_j has executed the protocol for $\gamma(n,m)$ rounds and that there exists $i \in \{1, \dots, n-(m+1)\}$ such that p_j belongs to the coalition $[i, \dots, i+m]$ or it belongs to the coalition $[i+1, \dots, i+m+1]$. Then after p_j has executed the protocol for $n-(m+1)$ more rounds, p_j belongs to the coalition $[i, \dots, i+m+1]$.*

Proof. We apply Lemma 7.3.3 in each round $\gamma(n,m)+c$, where $c \in \{1, \dots, n-(m+1)\}$. □

7.3.2 2Coalitions-consensus implements consensus

We are ready to give the proof of the last result that we need to obtain the main theorem of this chapter.

Theorem 7.3.5. *The protocol in Figure 7.3 implements the consensus task using $\binom{n}{2}$ (g, h)-2coalitions-consensus objects.*

Proof. The protocol clearly satisfies the Termination condition of the consensus task (the only loop is finite). We prove that it fulfills the Agreement and Validity conditions. Let p_j be any process that executes the protocol. Just at the beginning of the first round (line 2), process p_j belongs to the coalition $[j]$ (because of the assignments made to the local array $agreements_j$ in line 1, so that if p_j executes the protocol for $\gamma(n, 1) = n - 1$ rounds, we can conclude using Lemma 7.3.4, that process p_j belongs to some of the coalitions $[i, i+1]$ $(1 \leqslant i \leqslant n-1)$. Starting from this fact and using induction on m, we can prove that for all $m = 1, \ldots, n-1$; at the end of round $\gamma(n, m)$, p_j belongs to some of the coalitions $[i, \ldots, i+m]$ $(1 \leqslant i \leqslant n-m)$. In the last round (when $m = n-1$), after executing the main **for** block, process p_j belongs to the coalition $T = [1, \ldots, n]$, thus when p_j executes line 18, it will assign to the local variable dec_i a valid proposed input value and this is the value decided by p_j at line 19. All processes decide the same value because all of them are in the coalition T. Therefore the protocol satisfies the Agreement and Validity conditions of the consensus task. □

Replacing the call to the 2coalitions-consensus objects in the protocol of Figure 7.3 with the code of the protocol of g-2coalitions-consensus which is shown in Figure 7.1, we obtain a working iterated protocol for consensus using safe-consensus objects.

Theorem 7.3.6. *There exists an iterated protocol that solves the consensus task for n processes using $\binom{n}{2}$ safe-consensus objects.*

In the next chapters, we prove that this protocol is sharp on the number of safe-consensus objects invoked by the processes. We will show that it is impossible to implement consensus in the iterated model with safe-consensus objects using less that $\binom{n}{2}$ safe-consensus shared objects.

8 The structure of iterated protocols with safe-consensus

In this chapter, we present a series of results that describe the structure of executions of protocols in the iterated model with safe-consensus objects. The main objective here is to prove Theorem 8.2.5, which give us valuable information on the way two arbitrary successor states of a given reachable state in a iterated protocol with safe-consensus can be connected, by using a path of connected states which has very interesting properties related to the way in which the processes invoke the safe-consensus shared object in the protocol. To prove Theorem 8.2.5, we need to develop a very specific method to build the path connecting two given states of the protocol. We call this method the *Stairway method*, we describe it in full detail in Section 8.2 with many examples and the formal proofs of the related results. The Stairway method and Theorem 8.2.5 are fundamental to obtain the lower bound result of Chapter 9.

8.1 Preliminaries

Before proceeding to give the formal proofs of the results of this chapter, we need to introduce new definitions regarding the iterated model with safe-consensus objects. Some of these concepts will be used not only in this chapter, but also in the rest of this thesis.

8.1.1 Further terminology of the iterated model with safe-consensus

Remember from Chapter 6, that for $2 \leqslant m \leqslant n$ and a given protocol $\mathcal{A}$ in the iterated model with (safe-consensus) shared objects, a set $b \subseteq \overline{n}$ is an element of $\Gamma_{\mathcal{A}}(n, m)$ if and only if $|b| = m$ and there is at least an execution of $\mathcal{A}$ in which, in some round, only the processes with ids in the set b invoke a safe-consensus shared object. The symbol $\nu_{\mathcal{A}}(n, m)$ represents the cardinality of $\Gamma_{\mathcal{A}}(n, m)$ and we define the set $\Gamma_{\mathcal{A}}(n)$ and the quantity $\nu_{\mathcal{A}}(n)$ as follows: $\Gamma_{\mathcal{A}}(n) = \bigcup_{m=2}^{n} \Gamma_{\mathcal{A}}(n, m)$; $\nu_{\mathcal{A}}(n) = \sum_{i=2}^{n} \nu_{\mathcal{A}}(n, m)$. A set $b \in \Gamma_{\mathcal{A}}(n)$ representing a safe-consensus shared object invoked by some processes is called a *m-box* or simply a *box* ($m = |b|$). Notice that for the consensus protocol $\mathcal{A}$ of Chapter 7, we have that

$$\nu_{\mathcal{A}}(n, m) = n - m + 1 \text{ for } m \in \{2, \ldots, n\};$$

$$\nu_{\mathcal{A}}(n) = \binom{n}{2}.$$

We now introduce some technical definitions and ideas regarding properties of states, that will aid us to prove some lemmas that are needed for the proof of Theorem 8.2.5.

Definition 8.1.1. Let $\mathcal{A}$ be a protocol. For any path $\mathfrak{q}\colon Q_1 \sim \cdots \sim Q_l$ of connected states, define the *set of states* $\mathsf{States}(\mathfrak{q})$; the *set of indistinguishability sets* $\mathsf{iSets}(\mathfrak{q})$; and the *degree of indistinguishability* $\deg \mathfrak{q}$, of $\mathfrak{q}$ as follows:

$$\mathsf{States}(\mathfrak{q}) = \{Q_1, \ldots, Q_l\};$$

$$\mathsf{iSets}(\mathfrak{q}) = \{X \subseteq \overline{n} \mid (\exists Q_i, Q_j \in \mathsf{States}(\mathfrak{q}))(Q_i \overset{X}{\sim} Q_j)\};$$

$$\deg \mathfrak{q} = \min\{|X| \mid X \in \mathsf{iSets}(\mathfrak{q})\}.$$

The degree of indistinguishability of the path $\mathfrak{q}$ guarantees that we can find for any pair of states $Q_i, Q_j \in \mathsf{States}(\mathfrak{q})$ with $Q_i \sim Q_j$, a set of processes $P \subset \Pi$ that cannot distinguish between Q_i and Q_j of size at least $\deg \mathfrak{q}$. The degree of indistinguishability of a path is usually of non-importance in the standard and well known bivalency proofs of the impossibility of consensus in various systems [FLP85, LAA87], but in the main results of this chapter and also Chapter 9, measuring the degree of indistinguishability will be a recurring action in all the proofs.

Definition 8.1.2. Let $\mathcal{A}$ be an iterated protocol with safe-consensus objects and

$$S = \langle s_1, \ldots, s_n; SM; b_1, \ldots, b_q; o_1, \ldots, o_q \rangle$$

a reachable state in the protocol $\mathcal{A}$ (See Definition 6.1.5). The set of boxes $b_1, \ldots, b_q$ of the state S will be denoted by $\mathbf{Inv}(S)$, we call it the *invocation specification of* S. Let $b = b_j = \{l_1, \ldots, l_s\} \in \mathbf{Inv}(S)$, we define the *safe-consensus value of* b *in* S, denoted by $\mathsf{scval}(b, S)$ as the value o_j, that is, the output value of the safe-consensus shared object represented by b, which was invoked by the processes $p_{l_1}, \ldots, p_{l_s}$ to enter state S.

The following technical lemma is a result concerning invocation specifications of successor states of a given reachable state in an iterated protocol. It will be used later.

Lemma 8.1.3. *Let $\mathcal{A}$ be an iterated protocol for $n \geqslant 2$ processes with safe-consensus objects, S a reachable state in $\mathcal{A}$. Then for any two round schedules π_1, π_2, $\mathbf{Inv}(S \cdot \pi_1) = \mathbf{Inv}(S \cdot \pi_2)$.*

Proof. In Figure 6.1, notice that process p_i calls the object selector h with the local state it has from the previous round, so that if r is the round number of S and p_i starts executing the protocol $\mathcal{A}$ in round $r+1$ with the local state it had in S, the input to the object selector function h is given by the tuple $(r+1, i, sm_S, val_S)$, where sm_S, val_S depend only on S. Thus in both successor states $S \cdot \pi_1$ and $S \cdot \pi_2$, p_i feeds the same input to h, therefore

$$\mathbf{Inv}(S \cdot \pi_1) = \mathbf{Inv}(S \cdot \pi_2)$$

and the lemma is true. □

Definition 8.1.4. A path $\mathfrak{s}$ of connected states of $\mathcal{A}$ is said to be C-*regular* if and only if $\mathbf{Inv}(S) = \mathbf{Inv}(Q)$ for all $S, Q \in \mathsf{States}(\mathfrak{s})$.

In other words, $\mathfrak{s}$ is C-regular if and only if, all the states in the set $\mathsf{States}(\mathfrak{s})$ have the same global invocation specification. We will see examples of C-regular paths throughout the chapter.

A special round schedule

A basic building block of the proofs presented in this chapter and Chapter 9, is to define various ways in which the processes execute a protocol $\mathcal{A}$. For this action, we define a set of round schedules that will be very useful.

Let $\mathcal{A}$ be an iterated protocol, $q \geqslant 1$ and $A_1, \ldots, A_q \subset \overline{n}$ disjoint sets. Define the round schedule $\xi(A_1, \ldots, A_q, Y)$ for $\mathcal{A}$ as:

$$\mathsf{W}(A_1), \mathsf{S}(A_1), \mathsf{R}(A_1), \mathsf{W}(A_2), \mathsf{S}(A_2), \mathsf{R}(A_2), \ldots, \mathsf{W}(A_q), \mathsf{S}(A_q), \mathsf{R}(A_q), \mathsf{W}(Y), \mathsf{S}(Y), \mathsf{R}(Y), \quad (8.1)$$

where $Y = \overline{n} - (\bigcup_{i=1}^{q} A_i)$. When there is no confusion, we omit the set Y from the notation and just write $\xi(A_1, \ldots, A_q)$. Thus, in a given iteration of $\mathcal{A}$, the round schedule $\xi(A_1, \ldots, A_q, Y)$ just tell us that the processes with ids in the set A_1 perform the write-invoke safe-consensus-snapshot operations first (concurrently); then the processes with ids in A_2 do the same and so on until at the end, the processes with ids in A_q perform the main events of $\mathcal{A}$, followed by all the remaining processes. For any state S and $u \geqslant 0$, define the state

$$S \cdot \xi^u(A_1, \ldots, A_q) = \begin{cases} S & \text{if } u = 0, \\ (S \cdot \xi^{u-1}(A_1, \ldots, A_q)) \cdot \xi(A_1, \ldots, A_q) & \text{otherwise.} \end{cases}$$

I.e. $S \cdot \xi^u(A_1, \ldots, A_q)$ is the state that we obtain after we run the protocol $\mathcal{A}$ (starting from S) u rounds with the round schedule $\xi(A_1, \ldots, A_q)$ in each iteration.

8.2 The local structure of iterated protocols with safe-consensus: The Stairway method

We are ready to prove all the results of this chapter. We will prove three lemmas which are the basic building blocks to be able to show Theorem 8.2.5, these results describe (roughly) for a given iterated protocol, how two previously chosen reachable states can be connected in a very specific way, applying a construction process that we call the *Stairway method*. There are other simpler ways that we could use to prove there is a path between two chosen reachable states of an iterated protocol, and it would seem at first glance that using the Stairway method is unnecessarily difficult. But we have been able to prove Theorem 8.2.5 and the lower bound of the next chapter, only by applying the properties that can be obtained by using the Stairway method. We describe it in full detail in this section with the formal results that support it and also with many examples.

8.2.1 The construction of the Stairway method

The Stairway method is the key ingredient in the construction of a path of connected states (in a given iterated protocol) which will connect two previously chosen reachable states. The best way to describe the intuitive details of the method is by means of examples. Our strategy will be the following: We present with an example the problem that we want to solve with the Stairway method and then with a series of related examples, we explain the properties of the solution that we obtain with the Stairway method. Thus, our first example will be in fact, divided into four examples that will be presented throughout the rest of the chapter, together with the formal proof of the three lemmas that formalize the Stairway method.

The main example

Assume that $n = 7$ and that $\mathcal{A}$ is an iterated protocol with safe-consensus shared objects. We begin executing $\mathcal{A}$ until it enter some reachable state S. Let α be this partial execution of $\mathcal{A}$ which ends in the state S. Two possible one-round successor states of S are the following states

$$P = S \cdot \xi(\{1,2,3,4\},\{5,6,7\}) \quad \text{and} \quad Q = S \cdot \xi(\{5,6,7\},\{1,2,3,4\}), \tag{8.2}$$

that is, P is the successor state of S obtained when the processes p_1, p_2, p_3 and p_4 execute concurrently the write, invoke safe-consensus and snapshot operations, followed by p_5, p_6 and p_7 and to obtain the state Q, processes p_5, p_6, p_7 execute concurrently first, followed by p_1, p_2, p_3 and p_4. For the purposes of this example, assume also the following properties for P and Q:

$$\begin{gathered}\mathbf{Inv}(P) = \{b_1, b_2, b_3\}, \text{ where } b_1 = \{1,2\}, b_2 = \{3,4,5\} \text{ and } b_3 = \{5,6,7\};\\ \mathsf{scval}(b_1, P) = 2, \mathsf{scval}(b_2, P) = 3 \text{ and } \mathsf{scval}(b_3, P) = 7;\\ \mathsf{scval}(b_1, Q) = 2, \mathsf{scval}(b_2, Q) = 5 \text{ and } \mathsf{scval}(b_3, Q) = 7.\end{gathered} \tag{8.3}$$

Notice that by Lemma 8.1.3, $\mathbf{Inv}(P) = \mathbf{Inv}(Q)$. Our main goal is to connect the state P with Q with a path $\mathfrak{q}\colon P \sim \cdots \sim Q$. There are many simple ways to obtain a path $\mathfrak{q}$ between P and Q. For example, one possibility is the following path

$$\mathfrak{q}_1\colon P \overset{\{5,6,7\}}{\sim} S \cdot \xi(\overline{7}) \overset{\{1,2\}}{\sim} Q,$$

where (let $R = S \cdot \xi(\overline{7})$) $\mathbf{Inv}(R) = \mathbf{Inv}(P)$ (Lemma 8.1.3), $\mathsf{scval}(b_1, R) = 2, \mathsf{scval}(b_2, R) = 3$ and $\mathsf{scval}(b_3, R) = 7$. As P and R have the same safe-consensus values for the three boxes, the processes can only distinguish these states by the contents of the shared memory, thus p_5, p_6 and p_7 are the only processes that cannot distinguish between P and R. But we have that $\mathsf{scval}(b_2, R) \neq \mathsf{scval}(b_2, Q)$, thus only p_1, p_2 cannot distinguish between R and Q (notice that by the contents of the memory, only p_5, p_6, p_7 can distinguish between R and Q). The degree of indistinguishability of $\mathfrak{q}_1$ is $\deg \mathfrak{q}_1 = 2$. If we consider the possibility that $\mathsf{scval}(b_2, R) = 5$ (using the Safe-Validity of the safe-consensus task), then we obtain the path $\mathfrak{q}_2$ given by

$$\mathfrak{q}_2\colon P \overset{\{6,7\}}{\sim} R \overset{\{1,2,3,4\}}{\sim} Q,$$

so that only p_6, p_7 cannot distinguish between P and R, while the rest of the processes can distinguish either by the contents of the shared memory or because of the different safe-consensus values of the box b_2 in both states. We have also that $\deg \mathfrak{q}_2 = 2$.

Can we obtain a path between P and Q with a higher degree of indistinguishability ? The answer is yes, we can find a path $\mathfrak{q}$ such that $\deg \mathfrak{q} = 4 = 7 - |b_2|$. In the next examples, we build the path $\mathfrak{q}$ with the desired degree of indistinguishability and other useful properties, by applying the Stairway method.

Phase 1

In Lemma 8.2.2, we prove the first phase of the Stairway method. But before doing this, we present a short example of the results given in Lemma 8.2.2. We continue to develop the previous example. We have an iterated protocol $\mathcal{A}$ with safe-consensus objects, which is executed by the processes, accordingly to the specification of the partial execution α, until it enters into the reachable state S (Figure 8.1, we represent the boxes b_1, b_2, b_3 with colors). We try to connect the states P and Q with a path $\mathfrak{q}$, such that $\deg \mathfrak{q} = 7 - |b_2| = 4$. The first part of the path $\mathfrak{q}$ is constructed with phase one of the Stairway method, as follows: The partial execution α can be extended to a partial execution α_1, by applying the round schedule $\xi(\{1,2,3,4\},\{5,6,7\})$ in the next iteration of $\mathcal{A}$ to obtain the state P, as shown in Figure 8.1. A second partial execution α_2 can be obtained by extending α in

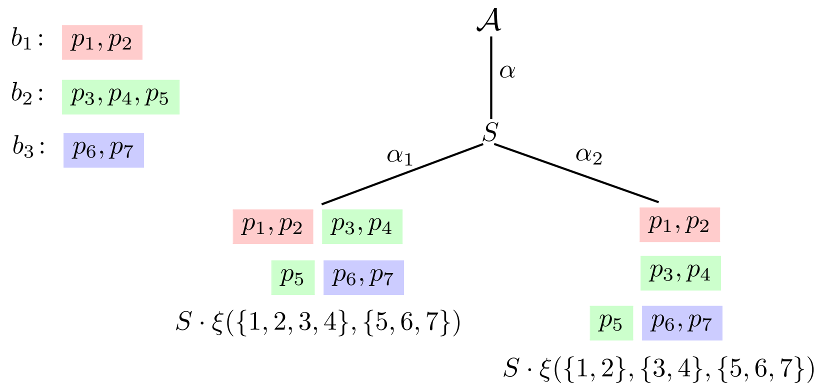


Figure 8.1: Phase one of the Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}$ and box b_2.

the next iteration by using the round schedule $\xi_1 = \xi(\{1,2\},\{3,4\},\{5,6,7\})$ so that the system enters the state $S_1 = S \cdot \xi_1$. Notice that by Lemma 8.1.3, $\mathbf{Inv}(P) = \mathbf{Inv}(S_1)$, thus the processes invoke the same safe-consensus shared objects in P and S_1. We choose the safe-consensus values of the boxes b_i in the state S_1 in the following way,

$$\mathsf{scval}(b_i, S_1) = \mathsf{scval}(b_i, P) \qquad \text{for all } i \in \{1,2,3\}. \tag{8.4}$$

In short, we want that the safe-consensus shared objects output the same values in both states P and S_1. This can be accomplished thanks to the Safe-Validity property of the safe-consensus task and the way the processes invoke the shared objects in P and S_1. For example, we know that $\mathsf{scval}(b_1, P) = 2$, then as both p_1 and p_2 execute concurrently the safe-consensus shared object represented by b_1 in the state S_1, then by the Safe-Validity condition, the output value of the shared object can be arbitrary. In particular, there exists the possibility that this output value is 2, so that we can make the safe-consensus value of b_1 in S_1 to be precisely 2, thus $\mathsf{scval}(b_1, P) = \mathsf{scval}(b_1, S_1)$. Similar arguments are used to prove the other equalities. Now, because all the safe-consensus objects invoked by the processes in P and S_1 output the same values on both states, the processes can distinguish between these states only by the contents of the shared memory. An easy analysis will show that P and S_1 are indistinguishable for p_3, p_4, p_5, p_6 and p_7 (p_1 and p_2 have a different view of the shared memory), thus the path $\mathfrak{p}_1$ connecting the states P and S_1 satisfies the inequality

$$\deg \mathfrak{p}_1 \geqslant 5 = 7 - 2.$$

This inequality is one of the main properties of $\mathfrak{p}_1$, and by Lemma 8.1.3, $\mathfrak{p}_1$ is a C-regular path. Also, notice how in the state S_1, the round schedule ξ_1 arranges the processes p_1, p_2, p_3 and p_4 by first executing the processes that belong to b_1, followed by the processes p_3, p_4 that invoke the object represented by b_2. We can say that these processes are placed in the levels of a small ladder, where the processes p_3, p_4, which invoke the safe-consensus object represented by b_2, are at the bottom level of the ladder, while the processes p_1, p_2, invoking the safe-consensus object given by b_1 take the top level. For this example, we have completed phase one of the stairway method.

We are ready to begin the construction of the Stairway method. The following definition plays an important role in describing the Stairway method.

Definition 8.2.1. Let $\mathcal{A}$ be a protocol with safe-consensus objects for n-processes, $X \subseteq \overline{n}$, R a state of some round $r > 0$ and $b \in \mathbf{Inv}(R)$. We say that R is a *ladder state for X with base b*, if $R = S \cdot \xi(C_1, \ldots, C_u, B)$ $(u \geqslant 0)$, where S is a reachable state in $\mathcal{A}$ and

- $X = (\bigcup_j^u C_j) \cup B$;
- for $j = 1, \ldots, u$, $0 \leqslant |C_j| \leqslant 2$;
- $(\bigcup_j^u C_j) \cap b = \varnothing$;
- $B = b \cap X$.

The state $S_1 = S \cdot \xi(\{1,2\},\{3,4\},\{5,6,7\}) = S \cdot \xi(\{1,2\},\{3,4\})$ of the previous example is a ladder state for $X = \{1,2,3,4\}$ with base b_2. For each box b_i and $W \subseteq \overline{n}$, let the set $W_{(i)}$ be defined as

$$W_{(i)} = W \cap b_i.$$

Lemma 8.2.2 (The Stairway method, Phase 1)**.** *Let $n \geqslant 2$, $\varnothing \neq X \subseteq \overline{n}$, $\mathcal{A}$ an iterated protocol with safe-consensus objects, S a state that is reachable in $\mathcal{A}$ in some round $r \geqslant 0$ and $b_j \in \mathbf{Inv}(S_X)$, where $S_X = S \cdot \xi(X)$. Then there exists a state L, such that L is a ladder state for X with base b_j, such that S_X and L are connected in round $r+1$ with a path of states $\mathfrak{p}$ with* $\deg \mathfrak{p} \geqslant n-2$.

Proof. Let $\mathbf{Inv}(S_X) = \{b_1, \ldots, b_q\}$ $(q \geqslant 1)$. By Lemma 8.1.3, for any one-round successor state Q of S, $\mathbf{Inv}(Q) = \mathbf{Inv}(S_X)$ and we can write $\mathbf{Inv}_S$ instead of $\mathbf{Inv}(S_X)$. Without loss, assume that $b_1 = b_j$. If $q = 1$, then $b_1 = \overline{n}$ and the result is immediate, because S_X is a ladder state for X with base $\overline{n}$. So that we suppose that $q > 1$. Partition X as $X = X_{(1)} \cup \cdots \cup X_{(q)}$ and build the following path of connected states

$$\begin{aligned} S_X \overset{\overline{n}-\Lambda_2(1)}{\sim} S \cdot \xi(\Lambda_2(1), X - \Lambda_2(1)) \overset{\overline{n}-\Lambda_2(2)}{\sim} S \cdot \xi(\Lambda_2(1), \Lambda_2(2), X - (\Lambda_2(1) \cup \Lambda_2(2))) \\ \overset{\overline{n}-\Lambda_2(3)}{\sim} \cdots \overset{\overline{n}-\Lambda_2(\alpha_2)}{\sim} \qquad\qquad\qquad \\ S \cdot \xi(\Lambda_2(1), \Lambda_2(2), \ldots, \Lambda_2(\alpha_2), X - X_{(2)}), \quad \boxed{8.5} \end{aligned}$$

where $X_{(2)} = \bigcup_{i=1}^{\alpha_2} \Lambda_2(i)$ is a partition of $X_{(2)}$ such that $|\Lambda_2(j)| = 1$ for $j = 2, \ldots, \alpha_2$. The set $\Lambda_2(1)$ has cardinality given by

$$|\Lambda_2(1)| = \begin{cases} 2 & \text{if } |X_{(2)}| > 1, \\ |X_{(2)}| & \text{otherwise,} \end{cases}$$

and we choose the safe-consensus value of every box in the set $\mathbf{Inv}_S$ to be the same in each state of the previous path. This can be done because of the way we partition $X_{(2)}$, the election of the elements of the set $\Lambda_2(1)$ and the properties of the safe-consensus task (Safe-Validity). We execute similar steps with box b_3, so that we obtain the path

$$\begin{aligned}
&S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_2(\alpha_2), X - X_{(2)}) \\
&\qquad \overset{\overline{n}-\Lambda_3(1)}{\sim} S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_2(\alpha_2), \Lambda_3(1), X - (X_{(2)} \cup \Lambda_3(1))) \\
&\qquad \overset{\overline{n}-\Lambda_3(2)}{\sim} S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_2(\alpha_2), \Lambda_3(1), \Lambda_3(2), X - (X_{(2)} \cup \Lambda_3(1) \cup \Lambda_3(2))) \\
&\qquad \overset{\overline{n}-\Lambda_3(3)}{\sim} \cdots \overset{\overline{n}-\Lambda_3(\alpha_3)}{\sim} \\
&\qquad S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_2(\alpha_2), \Lambda_3(1), \ldots, \Lambda_3(\alpha_3), X - (X_{(2)} \cup X_{(3)})), \qquad \boxed{8.6}
\end{aligned}$$

where the $\Lambda_3(i)$'s and α_3 depend on b_3 and $X_{(3)}$, just in the same way the sets $\Lambda_2(j)$ and α_2 depend on b_2 and $X_{(2)}$ - and each box has safe-consensus value equal to the value it has in the path of $\boxed{8.5}$. We can repeat the very same steps for $b_4, \ldots, b_q$ to obtain the path

$$\begin{aligned}
&S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_3(\alpha_3), X - (X_{(2)} \cup X_{(3)})) \\
&\qquad \overset{\overline{n}-\Lambda_4(1)}{\sim} S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_3(\alpha_3), \Lambda_4(1), X - (X_{(2)} \cup X_{(3)} \cup \Lambda_4(1))) \\
&\qquad \overset{\overline{n}-\Lambda_4(2)}{\sim} \cdots \overset{\overline{n}-\Lambda_q(1)}{\sim} \\
&\qquad S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_q(1), X - (\Lambda_q(1) \cup \bigcup_{i=2}^{q-1} X_{(i)})) \\
&\qquad \overset{\overline{n}-\Lambda_q(2)}{\sim} \cdots \overset{\overline{n}-\Lambda_q(\alpha_q)}{\sim} S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_q(\alpha_q), X_{(1)}). \qquad \boxed{8.7}
\end{aligned}$$

It is easy to prove that $L = S \cdot \xi(\Lambda_2(1), \ldots, \Lambda_q(\alpha_q), X_{(1)})$ is a ladder state for X with base b_1 and that each of the paths of equations $\boxed{8.5}$, $\boxed{8.6}$ and $\boxed{8.7}$ has indistinguishability degree no less that $n-2$. Combining all these paths, we obtain a new path

$$\mathfrak{p} \colon S_X \sim \cdots \sim L$$

with $\deg \mathfrak{p} \geqslant n-2$. □

Phase 2

Lemma 8.2.3 presents formally the second phase of the Stairway method. Now we continue to develop the example in which we presented phase one. Remember that the main goal of the example is to connect the two states P, Q of Equation $\boxed{8.2}$ of a reachable state S in an iterated protocol $\mathcal{A}$ with safe-consensus objects. The path connecting P and Q must have the highest possible degree of indistinguishability, which is $7 - |b_2| = 4$, where $b_2 \in \mathbf{Inv}(P)$ and $b_2 = \{3, 4, 5\}$ (see Equation $\boxed{8.3}$). In our first example, we built a path which connects the state P and the state $S_1 = S \cdot \xi(\{1,2\}, \{3,4\}, \{5,6,7\})$, which is a ladder state for $X = \{1,2,3,4\}$ with base $b_2 = \{3,4,5\}$ (Definition 8.2.1). The purpose of the second phase of the Stairway method, is to interchange the processes p_3, p_4 with process p_5 in the state S_1, formally, this is done in the following way: We extend the path $\mathfrak{p}_1$ of our first example with a path $\mathfrak{p}_2$, that connects S_1 with the state $S_3 = S \cdot \xi(\{1,2\}, \{5\}, \{3,4,6,7\})$ (Figure 8.2). Notice that by Definition 8.2.1, S_3 is a ladder state for $X' = \{1,2,5\}$ with base b_2.

In the example of phase one, we extended the partial execution α of $\mathcal{A}$ (that ends in the state S), to a partial execution α_2 using the round schedule $\xi(\{1,2\}, \{3,4\}, \{5,6,7\})$ to obtain the state S_1. Consider the two round schedules ξ', ξ'' given as

$\xi' = \xi(\{1,2\},\{3,4,5,6,7\}),$

$\xi'' = \xi(\{1,2\},\{5\},\{3,4,6,7\}).$

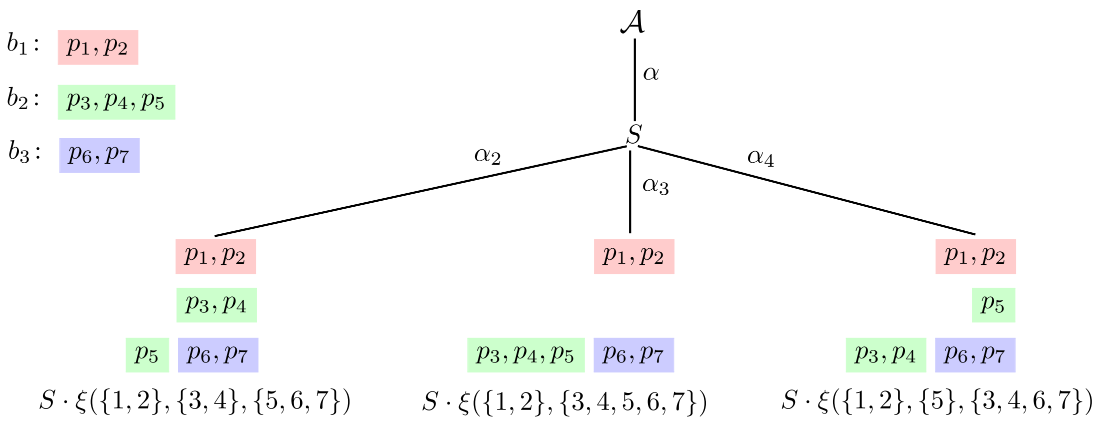


Figure 8.2: Phase two of the Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and b_2.

We use ξ' and ξ'' to extend α to new partial executions α_3, α_4 by using ξ' to extend α to α_3 with the state $S' = S \cdot \xi'$ and with ξ'' we obtain α_4 with the state $S'' = S \cdot \xi''$, see Figure 8.2. Notice that the difference between the states S_1 and S', is only that the processes p_3, p_4 executed in S_1 faster that in S', as in the later state, p_3 and p_4 execute concurrently together with p_5, p_6, p_7. From S' to S'', the difference is that we "lifted" process p_5, making it execute faster that p_3, p_4, p_6, p_7. We claim that the following C-regular path of connected states exists.

$$\mathfrak{p}_2 \colon S_1 \overset{\overline{7}-\{3,4\}}{\sim} S' \overset{\overline{7}-Z}{\sim} S''.$$

where the set $Z \subseteq \overline{7}$ is defined as

$$Z = \begin{cases} b_2 & \text{if } \mathsf{scval}(b_2, S') \neq \mathsf{scval}(b_2, S'') \\ \{5\} & \text{otherwise.} \end{cases} \qquad \boxed{8.8}$$

We first remark that by Lemma 8.1.3, $\mathbf{Inv}(S_1) = \mathbf{Inv}(S') = \mathbf{Inv}(S'')$, thus $\mathfrak{p}_2$ is indeed a C-regular path. Second, we can make the equalities $\mathsf{scval}(b_i, S_1) = \mathsf{scval}(b_i, S')$ hold true for all $i \in \{1,2,3\}$. This is done by using the Safe-Validity property of the safe-consensus task and the way the processes invoke the shared objects (either there are at least two processes invoking a safe-consensus object or the processes access a shared object in the same way in both states), so that the output values of the safe-consensus objects cannot help the processes to distinguish between S_1 and S'. The only two processes with a different view of the shared memory in S_1 and S' are p_3 and p_4, thus $S_1 \overset{\overline{7}-\{3,4\}}{\sim} S'$. For the case of the states S' and S'', notice that while in S', p_3 and p_4 invoked concurrently the safe-consensus object, in S'' p_5 is the first process to invoke the safe-consensus object and it outputs before p_3, p_4 access the object (Figure 8.2), hence by the Safe-Validity condition, we have that

$$\mathsf{scval}(b_2, S'') = 5$$

and for our examples, we have that $\mathsf{scval}(b_2, S') = 3 \neq 5$. This is where the cases of Equation $\boxed{8.8}$ come into play. If it was true that $\mathsf{scval}(b_2, S'') = \mathsf{scval}(b_2, S')$, then the processes could distinguish S' and S'' only by the contents of the shared memory and the only process with a different view of memory would be p_5, thus $Z = \{5\}$. But for our example, we have that $\mathsf{scval}(b_2, S'') \neq \mathsf{scval}(b_2, S')$, then not only p_5 has a different view of the shared memory in S' and S'', but also all the processes that invoke the shared object represented by b_2 see a different safe-consensus value in S' and S'' and only these processes can see a difference, thus $Z = b_2$ and we conclude that $S' \overset{\overline{7}-b_2}{\sim} S''$. We have proven that the path $\mathfrak{p}_2$ does exist and it satisfies the property

$$\deg \mathfrak{p}_2 = \begin{cases} 7-2 & \text{if } \mathsf{scval}(b_2, S') = \mathsf{scval}(b_2, S'') \\ 7 - |b_2| & \text{otherwise.} \end{cases}$$

In the second case of the previous argument, when $\mathsf{scval}(b_2, S'') \neq \mathsf{scval}(b_2, S')$, we have that $\overline{7} - b_2 \in \mathsf{iSets}(\mathfrak{p}_2)$. This says that the complement of a box, representing a safe-consensus shared object invoked by p_3, p_4, p_5, can be made the set of ids of processes that cannot distinguish between two states of the path $\mathfrak{p}_2$ (in this case, between S' and S''). This property plays an important role to prove the main results of the next chapter and it is a consequence of the Stairway method. We are now ready to prove the correctness of the second phase of the Stairway method for the general case.

Lemma 8.2.3 (The Stairway method, Phase 2). *Let $n \geqslant 2, X, Y \subseteq \overline{n}$, $\mathcal{A}$ an iterated protocol with safe-consensus objects and S an state that is reachable in $\mathcal{A}$ in some round $r \geqslant 0$. Assume also that b_j is a box representing a safe-consensus object used by some processes such that $b_j \in \mathbf{Inv}(L_1) = \mathbf{Inv}(L_2)$, where $L_1 = S \cdot \xi(C_1, \ldots, C_u, X_{(j)})$ is a ladder state for X with base b_j and $L_2 = S \cdot \xi(C_1, \ldots, C_u, Y_{(j)})$ is a ladder state for $(X - X_{(j)}) \cup Y_{(j)}$ with base b_j. Finally, suppose that if $b_j = \overline{n}$, $\mathsf{scval}(b_j, L_1) = \mathsf{scval}(b_j, L_2)$. Then L_1 and L_2 are connected in round $r+1$ with a C-regular path of states $\mathfrak{p}$ that satisfies the following properties:*

i) If $\mathsf{scval}(b_j, L_1) = \mathsf{scval}(b_j, L_2)$ then $\deg \mathfrak{p} \geqslant n-2$.

ii) If $\mathsf{scval}(b_j, L_1) \neq \mathsf{scval}(b_j, L_2)$ then $\deg \mathfrak{p} \geqslant n - |b_j|$ and $\overline{n} - b_j \in \mathsf{iSets}(\mathfrak{p})$.

Proof. Let $\overline{n}$ be the disjoint union $\overline{n} = X^- \cup (X \cap Y) \cup Y^- \cup W$, where

$X^- = X - Y;$

$Y^- = Y - X;$

$W = \overline{n} - (X^- \cup (X \cap Y) \cup Y^-).$

What we need to do to go from L_1 to L_2 is to "interchange" $X_{(j)}$ with $Y_{(j)}$. We first construct a path of connected states $\mathfrak{p}_1$ given by

$$
\begin{aligned}
& S \cdot \xi(C_1, \ldots, C_u, X_{(j)}, Y^- \cup W) \\
& \qquad \overset{\overline{n}-\Lambda_j(1)}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Lambda_j(1), X_{(j)} - \Lambda_j(1), Y^- \cup W) \\
& \qquad \overset{\overline{n}-\Lambda_j(2)}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Lambda_j(1), \Lambda_j(2), X_{(j)} - (\Lambda_j(1) \cup \Lambda_j(2)), Y^- \cup W) \\
& \qquad \overset{\overline{n}-\Lambda_j(3)}{\sim} \cdots \overset{\overline{n}-\Lambda_j(\alpha_j-1)}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Lambda_j(1), \Lambda_j(2), \ldots, \Lambda_j(\alpha_j), Y^- \cup W) \\
& \qquad \overset{\overline{n}-\Lambda_j(\alpha_j)}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Lambda_j(1), \Lambda_j(2), \ldots, \Lambda_j(\alpha_j - 1), \Lambda_j(\alpha_j) \cup Y^- \cup W) \\
& \qquad \overset{\overline{n}-\Lambda_j(\alpha_j-1)}{\sim} \cdots \overset{\overline{n}-\Lambda_j(1)}{\sim} S \cdot \xi(C_1, \ldots, C_u, X_{(j)} \cup Y^- \cup W),
\end{aligned}
$$

where the following properties hold:

- $X_{(j)} = \bigcup_{i=1}^{\alpha_j} \Lambda_j(i)$ is a partition of $X_{(j)}$ such that $|\Lambda_j(l)| = 1$ for $l = 2, \ldots, \alpha_j$;
- $|\Lambda_j(1)| = 2$ if $|X_{(j)}| > 1$ and $|\Lambda_j(1)| = |X_{(j)}|$ otherwise;
- $\mathfrak{p}_1$ is a C-regular path (Lemma 8.1.3) with $\deg \mathfrak{p}_1 \geqslant n - 2$;
- the safe-consensus value of every box b_i is the same in each state of $\mathfrak{p}_1$. This can be archived by a proper election of elements of the set $\Lambda_j(1)$ and the Validity property of the safe-consensus task.

Now, as $X_{(j)} \cup Y^- = Y_{(j)} \cup X^-_{(j)} \cup (Y^- - Y^-_{(j)})$, we can write the state $S \cdot \xi(C_1, \ldots, C_u, X_{(j)} \cup Y^- \cup W)$ as $S \cdot \xi(C_1, \ldots, C_u, Y_{(j)} \cup X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W)$. We need to build a second path $\mathfrak{p}_2$ as follows:

$$
\begin{aligned}
& S \cdot \xi(C_1, \ldots, C_u, Y_{(j)} \cup X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W) \\
& \qquad \overset{\overline{n}-Z}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Omega_j(1), (Y_{(j)} - \Omega_j(1)) \cup X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W) \\
& \qquad\qquad \overset{\overline{n}-\Omega_j(2)}{\sim} \cdots \overset{\overline{n}-\Omega_j(\epsilon_j-1)}{\sim} \\
& \qquad S \cdot \xi(C_1, \ldots, C_u, \Omega_j(1), \ldots, \Omega_j(\epsilon_j) \cup X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W) \\
& \qquad \overset{\overline{n}-\Omega_j(\epsilon_j)}{\sim} S \cdot \xi(C_1, \ldots, C_u, \Omega_j(1), \ldots, \Omega_j(\epsilon_j), X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W) \\
& \qquad\qquad \overset{\overline{n}-\Omega_j(\epsilon_j-1)}{\sim} \\
& \qquad S \cdot \xi(C_1, \ldots, C_u, \Omega_j(1), \ldots, \Omega_j(\epsilon_j - 1) \cup \Omega_j(\epsilon_j), X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W) \\
& \qquad \overset{\overline{n}-\Omega_j(\epsilon_j-2)}{\sim} \cdots \overset{\overline{n}-\Omega_j(1)}{\sim} S \cdot \xi(C_1, \ldots, C_u, Y_{(j)}, X^-_{(j)} \cup (Y^- - Y^-_{(j)}) \cup W).
\end{aligned}
$$

Let L_2 be the last state of the previous path. The next assertions are true for the path $\mathfrak{p}_2$:

- The sets $\Omega_j(i)$ and ϵ_j are defined for $Y_{(j)}$ and b_j in the same way as the $\Lambda_j(i)$ and α_j are defined for $X_{(j)}$ and b_j;
- The path $\mathfrak{p}_2$ is C-regular (Lemma 8.1.3);

- The safe-consensus value of every box $c \neq b_j$ is the same in every element of $\mathsf{States}(\mathfrak{p}_2)$;
- $\mathsf{scval}(b_j, Q) = \mathsf{scval}(b_j, P)$ for all $Q, P \in \mathsf{States}(\mathfrak{p}_2) - \{R\}$, where $R = S \cdot \xi(C_1, \ldots, C_u, Y_{(j)} \cup X^{-}_{(j)} \cup (Y^{-} - Y^{-}_{(j)}) \cup W)$;
- the set Z is defined by

$$Z = \begin{cases} b_j & \text{if } \mathsf{scval}(b_j, L_1) \neq \mathsf{scval}(b_j, L_2) \\ \Omega_j(1) & \text{otherwise;} \end{cases}$$

Notice that by the last assertion, we can deduce that $\deg \mathfrak{p}_2 \geqslant n - |b_j|$ and $\overline{n} - b_j \in \mathsf{iSets}(\mathfrak{p}_2)$ if $\mathsf{scval}(b_j, L_1) \neq \mathsf{scval}(b_j, L_2)$ and $\deg \mathfrak{p}_2 \geqslant n - 2$ when $\mathsf{scval}(b_j, L_1) = \mathsf{scval}(b_j, L_2)$. Thus we can use $\mathfrak{p}_1$ and $\mathfrak{p}_2$ to obtain a C-regular path $\mathfrak{p}$ which fulfills properties i)-ii) of the lemma and that concludes the proof. □

Phase 3

The final phase of the Stairway method is the third one and it is also the easiest phase of all three. From the two examples that we developed before the proofs of phases one and two (Lemmas 8.2.2 and 8.2.3), to be able to connect the two states P and Q of Equation $\boxed{8.2}$, successor states of a reachable state S in the iterated protocol $\mathcal{A}$, we have the following data:

A C-regular path $\mathfrak{p}_1$ connecting P and $S_1 = S \cdot \xi(\{1,2\},\{3,4\},\{5,6,7\})$;

A C-regular path $\mathfrak{p}_2$ connecting S_1 and $S'' = S \cdot \xi(\{1,2\},\{5\},\{3,4,6,7\})$ with the properties:

- If $\mathsf{scval}(b_2, S_1) = \mathsf{scval}(b_2, S'')$ then $\deg \mathfrak{p}' \geqslant 5 = 7 - 2$.
- If $\mathsf{scval}(b_2, S_1) \neq \mathsf{scval}(b_2, S'')$ then $\deg \mathfrak{p}' \geqslant 7 - |b_2|$ and $\overline{7} - b_2 \in \mathsf{iSets}(\mathfrak{p}')$.

(See the properties of P, Q and b_2 in Equation $\boxed{8.3}$). To finish the Stairway method for $X = \{1,2,3,4\}$ and $b_2 = \{3,4,5\}$, it only remains for us to connect the state S'' with the state $S_2 = S \cdot \xi(\{1,2,5\},\{3,4,6,7\})$ using a C-regular path $\mathfrak{p}_3$ with a high degree of indistinguishability, but this can be done by applying phase one of the Stairway method to the state S_2 and using S'' as the final ladder state. We can see the three paths $\mathfrak{p}_1, \mathfrak{p}_2$ and $\mathfrak{p}_2$ combined into a single path $\mathfrak{q}_1$ connecting the states P and S_2 in Figure 8.3. By the previous examples, this path has the following properties

- $\mathfrak{q}_1$ is a C-regular path.
- If $\mathsf{scval}(b_2, P) = \mathsf{scval}(b_2, S_2)$ then $\deg \mathfrak{q}_1 \geqslant 7 - 2$.
- If $\mathsf{scval}(b_2, P) \neq \mathsf{scval}(b_2, S_2)$ then $\deg \mathfrak{q}_1 \geqslant 7 - |b_2|$ and $\overline{7} - b_2 \in \mathsf{iSets}(\mathfrak{q}_1)$.

This finishes the Stairway method for the sets $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and the box $b_2 = \{3,4,5\}$. We have not been able to connect the states P and Q, we will do this in the final section of this chapter, after we prove Theorem 8.2.5, which tell us that to be able to connect P and Q, we only need to apply the Stairway method to the remaining boxes b_1 and b_3.

Here is the proof of the full Stairway method for an arbitrary iterated protocol and any pair of subsets $X, Y \subseteq \overline{n}$.

Lemma 8.2.4 (The complete Stairway method)**.** *Let $n \geqslant 2$, $X, Y \subseteq \overline{n}$, $\mathcal{A}$ an iterated protocol with safe-consensus objects, S a state that is reachable in $\mathcal{A}$ in some round $r \geqslant 0$ and b_j a box representing a safe-consensus object used by some processes in round $r+1$ such that $b_j \in \mathbf{Inv}(Q_1) = \mathbf{Inv}(Q_2)$, where $Q_1 = S \cdot \xi(X)$ and $Q_2 = S \cdot \xi(Y_{(j)} \cup (X - X_{(j)}))$. Assume also that if $b_j = \overline{n}$, $\mathsf{scval}(b_j, Q_1) = \mathsf{scval}(b_j, Q_2)$. Then the states Q_1 and Q_2 are connected in round $r+1$ with a C-regular path of states $\mathfrak{p}$ that satisfies the following properties:*

a) If $\mathsf{scval}(b_j, Q_1) = \mathsf{scval}(b_j, Q_2)$ then $\deg \mathfrak{p} \geqslant n - 2$.

b) If $\mathsf{scval}(b_j, Q_1) \neq \mathsf{scval}(b_j, Q_2)$ then $\deg \mathfrak{p} \geqslant n - |b_j|$ and $\overline{n} - b_j \in \mathsf{iSets}(\mathfrak{p})$.

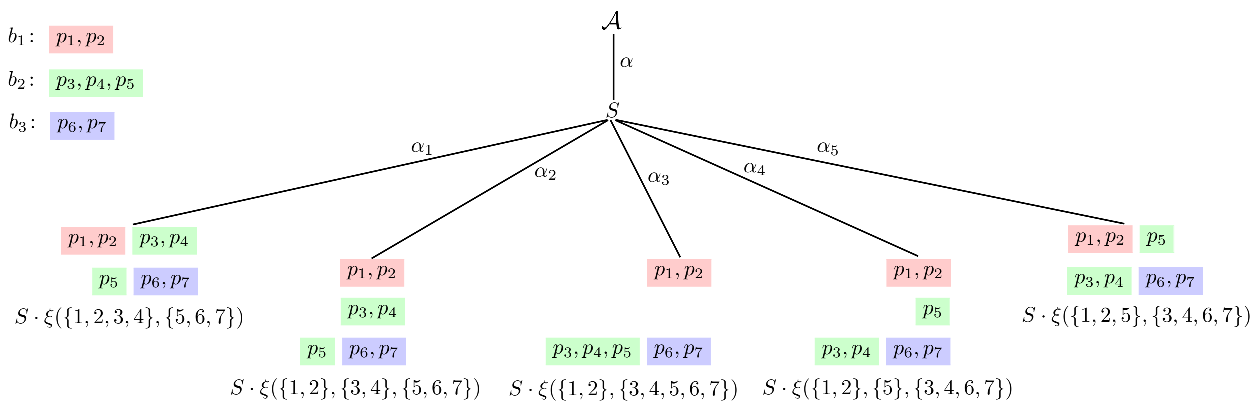

b_1: p_1, p_2
b_2: p_3, p_4, p_5
b_3: p_6, p_7
$\mathcal{A}$
α
S
α_1
α_2
α_3
α_4
α_5
$S \cdot \xi(\{1,2,3,4\},\{5,6,7\})$
$S \cdot \xi(\{1,2\},\{3,4\},\{5,6,7\})$
$S \cdot \xi(\{1,2\},\{3,4,5,6,7\})$
$S \cdot \xi(\{1,2\},\{5\},\{3,4,6,7\})$
$S \cdot \xi(\{1,2,5\},\{3,4,6,7\})$

Figure 8.3: The Stairway method for $n = 7$ processes, $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and b_2.

Proof. By Lemma 8.2.2, The state Q_1 can be connected with a state of the form $Q_{X_{(j)}} = S \cdot \xi(B_1, \ldots, B_s, X_{(j)})$ with a C-regular path $\mathfrak{q}_1$ such that $\deg \mathfrak{q}_1 \geqslant n - 2$. Using Lemma 8.2.3, $Q_{X_{(j)}}$ can be connected with $Q_{Y_{(j)}} = S \cdot \xi(B_1, \ldots, B_s, Y_{(j)})$ by means of a C-regular path $\mathfrak{q}_2 \colon Q_{X_{(j)}} \sim \cdots \sim Q_{Y_{(j)}}$ such that $\mathfrak{q}_2$ satisfies properties i) and ii) of that Lemma. And we can apply Lemma 8.2.2 to connect $Q_{Y_{(j)}}$ with Q_2 with the C-regular path $\mathfrak{q}_3$ which has indistinguishability degree no less that $n-2$. Therefore the path of connected states built by first gluing together the paths $\mathfrak{q}_1$ and $\mathfrak{q}_2$, followed by $\mathfrak{q}_3$, is a C-regular path $\mathfrak{q}$ such that the requirements a)-b) are satisfied. ☐

8.2.2 Local connectivity of iterated protocols with safe-consensus

In this section, we use the Stairway method of Lemma 8.2.4 to prove the main result of this chapter, Theorem 8.2.5. It will allow us to connect two given states of an iterated protocol with safe-consensus with a path $\mathfrak{q}$ of connected state with a high degree of indistinguishability and also the important property that the complements of some boxes representing safe-consensus shared objects of the protocol, belong to the set $\mathsf{iSets}(\mathfrak{q})$. After we have proven Theorem 8.2.5, we will be able to finally finish the example that we begun to develop at the beginning of Section 8.2.1 and it will help us to exemplify with specific data what is what can be done with Theorem 8.2.5.

Before proceeding to the proof of the main result of this chapter, we need one last technical definition. Let Q_1, Q_2 be two reachable states in round r of an iterated protocol $\mathcal{A}$ for n processes with safe-consensus objects. The set $\mathfrak{D}^r_{\mathcal{A}}(Q_1, Q_2)$ is defined as

$$\mathfrak{D}^r_{\mathcal{A}}(Q_1, Q_2) = \{b \in \Gamma_{\mathcal{A}}(n) \mid b \in \mathbf{Inv}(Q_1) \cap \mathbf{Inv}(Q_2) \text{ and } \mathsf{scval}(b, Q_1) \neq \mathsf{scval}(b, Q_2)\}. \qquad \boxed{8.9}$$

For our long lasting example and the states P, Q given in $\boxed{8.2}$ and accordingly to the properties of P and Q presented in $\boxed{8.3}$, we have that $\mathfrak{D}^r_{\mathcal{A}}(P, Q) = \{b_2\}$, where r is the round number of both P and Q.

Theorem 8.2.5. *Let $n \geqslant 2$ and $X, Y \subseteq \overline{n}$, $\mathcal{A}$ an iterated protocol with safe-consensus objects, S a reachable state of $\mathcal{A}$ in some round $r \geqslant 0$ and let $Q_1 = S \cdot \xi(X)$ and $Q_2 = S \cdot \xi(Y)$ be such that $\overline{n} \notin \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, Q_2)$. Then Q_1 and Q_2 are connected in round $r + 1$ with a C-regular path of states $\mathfrak{p}$ such that*

(A) If $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, Q_2) = \varnothing$, then $\deg \mathfrak{p} \geqslant n - 2$.

(B) If the set $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, Q_2)$ is not empty, then

1. *$\deg \mathfrak{p} \geqslant \min\{n - |b|\}_{b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, Q_2)}$;*
2. *for every $Z \in \mathsf{iSets}(\mathfrak{p})$ with $|Z| < n - 2$, there exists an unique $b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, Q_2)$ such that $Z = \overline{n} - b$.*

Proof. By Lemma 8.1.3, $\mathfrak{D}^r_{\mathcal{A}}(P_1, P_2) = \{b \in \Gamma_{\mathcal{A}}(n) \mid b \in \mathbf{Inv}_S \text{ and } \mathsf{scval}(b, P_1) \neq \mathsf{scval}(b, P_2)\}$, where P_l is a one-round successor state of S and $\mathbf{Inv}_S = \mathbf{Inv}(P_l)$, $l = 1, 2$. Let $\mathbf{Inv}_S = \{b_1, \ldots, b_q\}$. By Lemma 8.2.4, we can connect the state Q_1 with $R_1 = S \cdot \xi(Y_{(1)} \cup (X - X_{(1)}))$ using a C-regular path $\mathfrak{p}_1 \colon Q_1 \sim \cdots \sim R_1$ such that

(A_1) If $b_1 \notin \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_1)$, then $\deg \mathfrak{p}_1 \geqslant n - 2$.

(B_1) If $b_1 \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_1)$, then $\deg \mathfrak{p}_1 \geqslant n - |b_1|$ and $\overline{n} - b_1 \in \mathsf{iSets}(\mathfrak{p}_1)$.

Notice that if there is a set $Z \in \mathsf{iSets}(\mathfrak{p}_1)$ with size strictly less that $n - 2$, then it must be true that $b_1 \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_1)$, because if $b_1 \notin \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_1)$, the by (A_1), $|Z| \geqslant n - 2$ and this is impossible. Thus the conclusion of property (B_1) holds for $\mathfrak{p}_1$, which means that $\overline{n} - b_1 \in \mathsf{iSets}(\mathfrak{p}_1)$. Examining the proof of Lemma 8.2.4 we can convince ourselves that every $W \in \mathsf{iSets}(\mathfrak{p}_1)$ such that $W \neq \overline{n} - |b_1|$ has cardinality at least $n - 2$, thus $Z = \overline{n} - b_1$.

Applying Lemma 8.2.4 to the states R_1 and $R_2 = S \cdot \xi(Y_{(1)} \cup Y_{(2)} \cup (X - (X_{(1)} \cup X_{(2)})))$, we find a C-regular path $\mathfrak{p}_2 \colon R_1 \sim \cdots \sim R_2$ such that $\mathfrak{p}_2$ and b_2 enjoy the same properties which $\mathfrak{p}_1$ and b_1 have. We can combine the paths $\mathfrak{p}_1$ and $\mathfrak{p}_2$ to obtain a C-regular path $\mathfrak{p}_{12} \colon Q_1 \sim \cdots \sim R_2$ from Q_1 to R_2 satisfying the properties

(A_2) If $\{b_1, b_2\} \cap \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_2) = \varnothing$, then $\deg \mathfrak{p}_{12} \geqslant n - 2$.

(B_2) If $\{b_1, b_2\} \cap \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_2)$ is not empty, then

1. $\deg \mathfrak{p}_{12} \geqslant \min\{n - |b|\}_{b \in \{b_1, b_2\} \cap \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_2)}$;
2. for every $Z \in \mathsf{iSets}(\mathfrak{p}_{12})$ with $|Z| < n - 2$, there exists an unique $b \in \{b_1, b_2\} \cap \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_1, R_2)$ such that $Z = \overline{n} - b$.

We can repeat this process for all $s \in \{1,\dots,q\}$. In general, if $R_s = S\cdot\xi((\bigcup_i^s Y_{(i)}) \cup (X - (\bigcup_i^s X_{(i)})))$, we can construct a C-regular path

$$\mathfrak{p}_{1s} \colon Q_1 \sim \cdots \sim R_s,$$

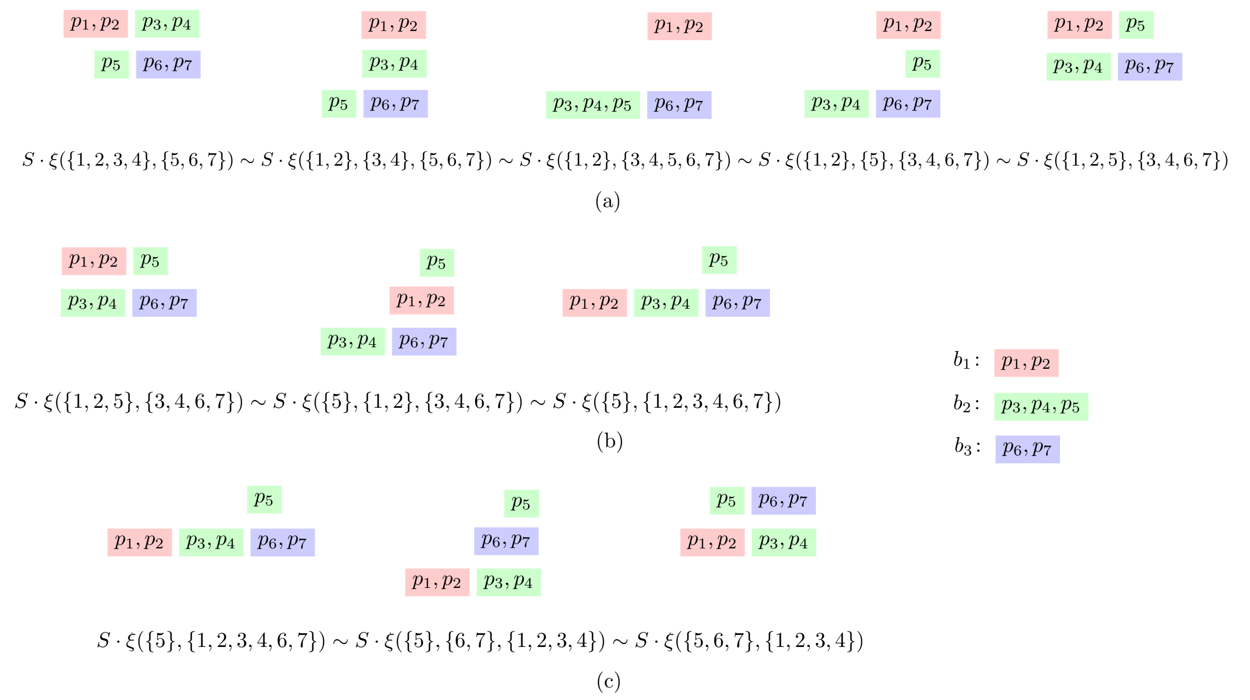
with the properties

(A_s) If $\{b_1,\dots,b_s\} \cap \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,R_s) = \varnothing$, then $\deg \mathfrak{p}_{1s} \geqslant n-2$.

(B_s) If $\{b_1,\dots,b_s\} \cap \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,R_s)$ is not empty, then

1. $\deg \mathfrak{p}_{1s} \geqslant \min\{n-|b|\}_{b\in\{b_1,\dots,b_s\}\cap\mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,R_s)}$;
2. for every $Z \in \mathsf{iSets}(\mathfrak{p}_{1s})$ with $|Z| < n-2$, there exists an unique $b \in \{b_1,\dots,b_s\} \cap \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,R_s)$ such that $Z = \overline{n} - b$.

As $R_q = S\cdot\xi((\bigcup_i^q Y_{(i)}) \cup (X - (\bigcup_i^q X_{(i)}))) = S\cdot\xi(Y) = Q_2$, the path $\mathfrak{p}_{1q}$ is the desired C-regular path from Q_1 to Q_2, fulfilling conditions (A) and (B) (because $\{b_1,\dots,b_q\} \cap \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,R_q) = \mathbf{Inv}_S \cap \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,Q_2) = \mathfrak{D}_{\mathcal{A}}^{r+1}(Q_1,Q_2)$). The result follows. $\square$



p_1,p_2 p_3,p_4 p_5 p_6,p_7
$S\cdot\xi(\{1,2,3,4\},\{5,6,7\}) \sim S\cdot\xi(\{1,2\},\{3,4\},\{5,6,7\}) \sim S\cdot\xi(\{1,2\},\{3,4,5,6,7\}) \sim S\cdot\xi(\{1,2\},\{5\},\{3,4,6,7\}) \sim S\cdot\xi(\{1,2,5\},\{3,4,6,7\})$
(a)
$S\cdot\xi(\{1,2,5\},\{3,4,6,7\}) \sim S\cdot\xi(\{5\},\{1,2\},\{3,4,6,7\}) \sim S\cdot\xi(\{5\},\{1,2,3,4,6,7\})$
b_1: p_1,p_2
b_2: p_3,p_4,p_5
b_3: p_6,p_7
(b)
$S\cdot\xi(\{5\},\{1,2,3,4,6,7\}) \sim S\cdot\xi(\{5\},\{6,7\},\{1,2,3,4\}) \sim S\cdot\xi(\{5,6,7\},\{1,2,3,4\})$
(c)

Figure 8.4: The path of the proof of Theorem 8.2.5, for $n = 7$ processes, the sets $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and the boxes b_1, b_2, b_3.

As an example of the result proved in Theorem 8.2.5, we connect the states P, Q presented in the examples of Section 8.2.1. By applying Theorem 8.2.5, we obtain a C-regular path $\mathfrak{q}$, which is given in Figure 8.4. It is divided into three parts for the sake of space. The first section of the path $\mathfrak{q}$ is the five-state path $\mathfrak{q}_1$ specified in part (a) of the figure, this is precisely the path that we constructed in the previous section using the Stairway method (Lemma 8.2.4) applied to the sets $X = \{1,2,3,4\}, Y = \{5,6,7\}$ and box $b_2 = \{3,4,5\}$. The middle path $\mathfrak{q}_2$ of part (b) of Figure 8.4, is obtained by using the Stairway method[1] for the sets $X' = \{1, 2, 5\}, Y' = \{5\}$ and the box $b_1 = \{1, 2\}$. The last path $\mathfrak{q}_3$ in part (c) of Figure 8.4 is obtained again with the Stairway method used for the sets $X'' = \{5\}, Y'' = \{5, 6, 7\}$ and box b_3. Joining the three paths we obtain the desired C-regular path $\mathfrak{q}$ connecting P and Q. For simplicity, we have omitted the elements of the set $\mathsf{iSets}(X)$ from the drawings of Figure 8.4, but it is easy to see that $\deg \mathfrak{q} \geqslant 4 = 7 - 3 = 7 - |b_2|$. Also, as $\mathsf{scval}(b_2, P) = 3 \neq 5 = \mathsf{scval}(b_2, Q)$, we have that $\{1, 2, 6, 7\} = \overline{7} - \{3, 4, 5\} = \overline{7} - b_2 \in \mathsf{iSets}(\mathfrak{q})$.

In the next chapter, we use Theorem 8.2.5 to prove the main result of the second part of this thesis: A matching lower bound on the number of safe-consensus shared objects needed to solve the consensus task with an iterated protocol with safe-consensus shared objects. Thus we will prove that the consensus protocol of Chapter 7 is sharp.

[1]Of course, it can happen that in a given situation, it is not necessary to apply all the steps of the Stairway method.

9 A Tight lower bound to implement consensus in the iterated model with safe-consensus

In this Chapter, we present the main result of the second part of this thesis: A matching lower bound on the number of safe-consensus objects needed to solve consensus with any iterated protocol, as we have defined them in Chapter 6. Our lower bound proof is based on standard bivalency arguments [FLP85], but in order to be able to apply them, a careful combinatorial work is necessary. We give a brief analysis and describe all the details in the following pages.

The combinatorial results which we develop in this chapter, tell us something about the connectivity of simple graphs, specifically, connectivity of subgraphs of *Johnson graphs*. In Section 9.1, we introduce the necessary concepts regarding graph theory and Johnson graphs and then we prove Theorem 9.1.10, one of the results that we need to be able to show the lower bound result. Later, in Section 9.2, we give an intuitive overview of the main details behind the proof of the lower bound. We use the consensus protocol of Chapter 7 to show how the processes execute the protocol and invoke the safe-consensus shared objects against a path of connected initial states, in such a way that in each iteration of the protocol, the path of initial states is transformed into a path with decreasing degree of indistinguishability, until the paths connecting states vanish and the processes can make a decision on the consensus output. We also modify this protocol to obtain a "bad protocol" with few safe-consensus objects and we show how this protocol is unable to solve consensus, because it cannot disconnect two specific reachable states, which are connected with a path. Thus, this bad protocol will provide us with an example exhibiting the key ingredients of the lower bound proof. In the last section, we present the formal proof of the lower bound on the number of safe-consensus objects needed by any iterated protocol with safe-consensus objects that solves consensus.

9.1 The combinatorics of iterated protocols with safe-consensus

In this section, we present definitions and results regarding graph theory and in particular, Johnson graphs and the connectivity of their subgraphs, which are needed to prove Theorem 9.1.10, the combinatorial result needed in the lower bound proof. We also gives a brief analysis of how our lower bound result was born, together with the formulation of Theorem 9.1.10. To do this analysis, we use the consensus protocol presented in Chapter 7 and examining its combinatorial properties.

9.1.1 Subgraphs of Johnson graphs

Our graph terminology is standard, see for example [Bol98], all the graphs that we use are simple graphs. For any set A and $E \subseteq \mathcal{P}(A)$, we denote the union of all the elements of every set in E as $\bigcup E$.

Definition 9.1.1. For $1 \leqslant m \leqslant n$, the *Johnson graph* $J_{n,m}$ has as vertex set all subsets of $\overline{n}$ of cardinality m, two vertices b_1, b_2 are adjacent if and only if $|b_1 \cap b_2| = m - 1$. The set $V(J_{n,m})$ will be denoted by $V_{n,m}$.

Johnson graphs are closely related to coding theory, in particular, they have a deep relationship with *Johnson schemes* [DL98]. Both Johnson graphs and Johnson schemes are named after Selmer M. Johnson. Examples of Johnson graphs include all complete graphs on n vertices, isomorphic to $J_{n,1}$; the *octahedral graph* is the graph $J_{4,2}$; $J_{5,2}$ is isomorphic to the complement of the *Petersen graph*.

Definition 9.1.2. Let $1 \leqslant m \leqslant n$ and $U \subseteq V_{n,m}$. Define the set $\zeta(U)$ as

$$\zeta(U) = \{c \cup d \mid c, d \in U \text{ and } |c \cap d| = m - 1\}. \tag{9.1}$$

Notice that each $f \in \zeta(U)$ has size $m + 1$ because, if $f = c \cup d$ for $c, d \in U$, then $|c| = |d| = m$ and it is known that $|c \cap d| = m - 1 \Leftrightarrow |c \cup d| = m + 1$. Thus $\zeta(U) \subseteq V_{n,m+1}$.

Definition 9.1.3. For any $U \subseteq V_{n,m}$ and $v = 0, \ldots, n - m$, the *iterated ζ-operator* ζ^v is given by

$$\zeta^v(U) = \begin{cases} U & \text{if } v = 0, \\ \zeta(\zeta^{v-1}(U)) & \text{otherwise.} \end{cases} \tag{9.2}$$

As $U \subseteq V_{n,m}$, we can check that $\zeta^v(U) \subseteq V_{n,m+v}$. A simple, but useful property of the ζ-operator is that

$$\bigcup \zeta^v(U) \subseteq \bigcup U. \tag{9.3}$$

9.1.2 Combinatorial properties of a consensus protocol

Before proving Theorem 9.1.10, we give a short description of how the main statement of the lower bound result and Theorem 9.1.10 were conceived. We use as a guiding example the consensus protocol of Chapter 7.

In view of the representation of invocations of safe-consensus shared objects as boxes (that is, as subsets of $\overline{n}$, see Definitions 6.1.3 and 6.1.4) in a protocol with safe-consensus, we can use the framework introduced in the previous section to identify combinatorial properties of iterated protocols with safe-consensus that solve the consensus task. Let $\mathcal{A}$ be the consensus protocol of Chapter 7 and suppose that $n = 4$. It is easy to see that for all $m \in \{2, 3, 4\}$,

$$\nu_{\mathcal{A}}(4, m) = 4 - m + 1 \qquad \text{and} \qquad \nu_{\mathcal{A}}(4) = \binom{4}{2} = 6.$$

In part (a) of Figure 9.1, we can see the same figure that we studied for the first time in Section 7.2. This is a graphical representation of the invocations of shared object performed by the processes when they execute the protocol $\mathcal{A}$ to solve consensus, for the case of four processes. By the results of Chapter 7, we can identify a 2coalitions-consensus object 2CC$_i$ with a safe-consensus shared object and each safe-consensus object can be identified with a box b_i, accordingly to which processes are invoking the shared object. This new representation of the execution of $\mathcal{A}$ using boxes is depicted in part (b) of Figure 9.1, so that we have that

$\Gamma_{\mathcal{A}}(4,2) = \{\{1,2\},\{2,3\},\{3,4\}\};$

$\Gamma_{\mathcal{A}}(4,3) = \{\{1,2,3\},\{2,3,4\}\};$

$\Gamma_{\mathcal{A}}(4,4) = \{\overline{4}\}.$

It is not hard to see that if we modify the protocol $\mathcal{A}$ in such a way that the processes p_3, p_4 never invoke a safe-consensus object only the two of them, then the modified protocol will fail to solve the consensus task and the same is true if the processes p_2, p_3 never invoke a safe-consensus object, or any other choice of removal of invocations of safe-consensus objects done by the processes in $\mathcal{A}$ and represented in Figure 9.1.

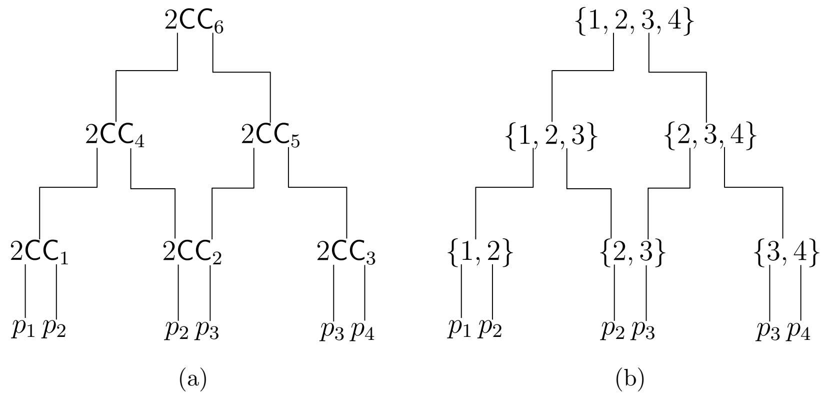


Figure 9.1: The combinatorial representation of the invocations of safe-consensus objects in a 4-process consensus protocol.

The question that we must try to answer is this: What happens combinatorially when we remove one safe-consensus invocation (i.e., a box) from $\mathcal{A}$? In order to give an answer, we can use the framework of Johnson graphs previously defined.

Let $U_m = \Gamma_{\mathcal{A}}(4,m)$ $(m \in \{2,3,4\})$. We can identify each set U_m of $\mathcal{A}$ as a subset of vertices of $J_{4,m}$, thus, $U_m \subset V_{4,m}$. U_m induces a subgraph G_m of $J_{4,m}$, $m \in \{2,3,4\}$. Now, Figure 9.1 suggest that, in some manner, the processes obtain the set of 3-boxes U_3 by applying the ζ-operator to the set U_2; and the singleton U_4 is obtained with one application of ζ to U_3. In short,

$$\zeta(U_2) = U_3 \quad \text{and} \quad \zeta(U_3) = U_4. \tag{9.4}$$

But using the two equalities with the iterated ζ-operator we have also that

$$\zeta^2(U_2) = \zeta(\zeta(U_2)) = \zeta(U_3) = U_4, \tag{9.5}$$

therefore, $\zeta^2(U_2) = U_4 = \{\overline{4}\}$ and the consensus protocol of Chapter 7 satisfies these equalities with the ζ-operator and the sets of boxes U_m. Notice also that the graphs G_m, induced by the boxes of $\mathcal{A}$, are all connected and each G_m has order $4 - m + 1$.

What happens if we remove one vertex, for example, $b = \{3,4\}$ from the graph G_2 ? Or equivalently, what happens if we forbid processes p_3, p_4 to invoke together a safe-consensus object in $\mathcal{A}$? In this case, we obtain a new subgraph G_2' of $J_{4,2}$, induced by the set of vertices $U_2' =$

$\{\{1,2\},\{2,3\}\}$ of cardinality $4-2=2$. The equalities of $\boxed{9.4}$ and $\boxed{9.5}$ are no longer true for the sets U_2', U_3 and U_4, as

$$\zeta(U_2') = \{\{1,2,3\}\} \subset U_3 \quad \text{and} \quad \zeta^2(U_2') = \zeta(\{\{1,2,3\}\}) = \varnothing \neq U_4.$$

If instead of considering the graph induced by U_2', we consider the graph G_2'' induced by the set $U_2'' = \{\{1,2\},\{3,4\}\}$ (again, we forbid the invocations of safe-consensus objects represented by the box $\{2,3\}$), then the result is basically the same, because $\zeta(U_2'') = \varnothing$, so that $\zeta^2(U_2'') = \varnothing \neq \Gamma_{\mathcal{A}}(4,4)$. Notice also that while G_2' is connected, G_2'' is disconnected, but both graphs have only $4-2=2$ vertices. We can generate similar results if we remove one vertex from any of the sets U_3, U_4.

All these examples suggest that for any iterated protocol for four processes that solves the consensus task, the equalities given in $\boxed{9.4}$ and $\boxed{9.5}$ must be fulfilled and a necessary condition for this to be true is that $\nu_{\mathcal{A}}(4,m) > 4-m$ for all $m = 2,3,4$. With a little effort, we can repeat this experiment for the consensus protocol of Chapter 7 for any $n > 4$ and the results would be the same. In summary, these examples lead us to conjecture that for any $n \geqslant 2$ and any iterated protocol $\mathcal{A}$ that solves n-consensus, it must be true that

$$\nu_{\mathcal{A}}(n,m) > n-m \qquad \text{for all } m \in \{2,\ldots,n\},$$

that is, the protocol $\mathcal{A}$ must allow the processes to invoke safe-consensus shared objects by sets of m processes in at least $n-m+1$ different ways. Thus, in order to show true our lower bound result, our main objective is to prove that if $\nu_{\mathcal{A}}(n,m) \leqslant n-m$ for some m, then $\mathcal{A}$ cannot solve the consensus task. And what happens in general to the combinatorics of $\mathcal{A}$ when $\nu_{\mathcal{A}}(n,m) \leqslant n-m$? Our 4-process example says that $\zeta^2(U_2') = \varnothing$, which can be interpreted in the following way: It is impossible to obtain the set $\Gamma_{\mathcal{A}}(4,4) = \{\overline{4}\}$ if the number of 2-boxes is at most $4-2=2$. This can be expressed in the general setting as follows:

$$\text{If } \nu_{\mathcal{A}}(n,m) \leqslant n-m, \text{ then } \zeta^{n-m}(\Gamma_{\mathcal{A}}(n,m)) = \varnothing.$$

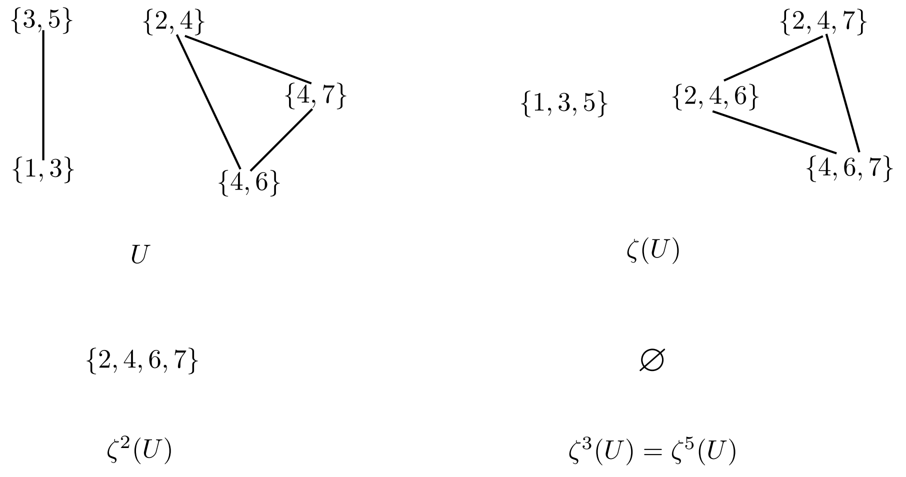


Figure 9.2: A subgraph of $J_{7,2}$ with $7-2=5$ vertices such that $\zeta^{7-2}(U) = \varnothing$.

But this is nothing more that an application of Theorem 9.1.10. We will take any subset $U \subseteq V_{n,m}$ such that $|U| \leqslant n-m$ and prove that $\zeta^{n-m}(U) = \varnothing$. We can see a graphical example of

Theorem 9.1.10 in Figure 9.2. In Section 9.3, we will confirm that this combinatorial property is indeed necessary to solve n-consensus with any iterated protocol with safe-consensus objects.

9.1.3 The connectivity of subgraphs of $J_{n,m}$

We are ready to prove all the combinatorial results that we need. Besides the inequalities with respect to the quantity $n-m$, we will see in the results of this section that connectivity also plays an important role in the development of Theorem 9.1.10.

Lemma 9.1.4. *Let $U \subseteq V_{n,m}$ and $G = G\left[U\right]$. The following properties are satisfied.*

(i) If G is connected, then $\left|\bigcup U\right| \leqslant m-1+|U|$.

(ii) If $U_1, \ldots, U_r$ are connected components of G, then $\zeta(\bigcup_{i=1}^{r} U_i) = \bigcup_{i=1}^{r} \zeta(U_i)$.

Proof. (i) can be proven easily using induction on $|U|$ and (ii) is an easy consequence of the definitions, thus we omit the proof. □

Lemma 9.1.5. *Let $U \subseteq V_{n,m}$ with $|U| \leqslant n-m$. If $G\left[U\right]$ is connected then $\bigcup U \neq \overline{n}$.*

Proof. Suppose that with the given hypothesis $\bigcup U = \overline{n}$. As $G\left[U\right]$ is connected, we can use part (i) of Lemma 9.1.4 to obtain the inequality $n \leqslant m + |U| - 1$. But this implies that $|U| \geqslant n-m+1$ and this is a contradiction. Therefore $\bigcup U \neq \overline{n}$. □

The following lemma is about subgraphs of $J_{n,2}$, it is a small result needed to give the full proof of our lower bound (see Theorem 9.3.5). However, this result is far easier to prove than Theorem 9.1.10.

Lemma 9.1.6. *Let $U \subset V_{n,2}$ with $|U| \leqslant n-2$. Then there exists a partition $\overline{n} = A \cup B$ such that*

$$(\forall b \in U)(b \subseteq A \text{ or } b \subseteq B).$$

Proof. If $\bigcup U \neq \overline{n}$, then setting $A = \overline{n} - \{i\}$ and $B = \{i\}$ where $i \notin \bigcup U$ we are done. Otherwise, by Lemma 9.1.5, the induced subgraph $G = G\left[U\right]$ of $J_{n,2}$ is disconnected. There exists a partition V_1, V_2 of $V(G) = U$ with the property that there is no edge of G from any vertex of V_1 to any vertex of V_2. Let

$$A = \bigcup V_1 \quad \text{and} \quad B = \bigcup V_2.$$

It is easy to show that $\overline{n} = A \cup B$ is the partition of $\overline{n}$ that we need. □

Lemma 9.1.7. *Let $U \subset V_{n,m}$ with $|U| > 1$ and $G = G\left[U\right]$ a connected subgraph of $J_{n,m}$. Then the graph $G' = G\left[\zeta(U)\right]$ is connected and $\bigcup \zeta(U) = \bigcup U$.*

Proof. We show that G' is connected. If G contains only two vertices, then G' contains only one vertex and the result is immediate. So assume that $|U| > 2$ and take $b, c \in \zeta(U)$. We need to show that there is a path from b to c. We know that $b = b_1 \cup b_2$ and $c = c_1 \cup c_2$ with $b_i, c_i \in U$ for $i = 1, 2$. As G is connected, there is a path

$$v_1 = b_2, v_2, \ldots, v_q = c_2,$$

and we use it to build the following path in G',

$$b = b_1 \cup v_1, v_1 \cup v_2, \ldots, v_{q-1} \cup v_q, c = v_q \cup c_1,$$

thus b and c are connected in G', so that it is a connected graph.

Now we prove that $\bigcup \zeta(U) = \bigcup U$. By Equation $\boxed{9.3}$, $\bigcup \zeta(U) \subseteq \bigcup U$, so it only remains to prove the other inclusion. Let $x \in \bigcup U$, there is a vertex $b \in U$ such that $x \in b$ and as $|U| > 1$ and G is connected, there exists another vertex $c \in U$ such that b and c are adjacent in G. Then $b \cup c \in \zeta(U)$, thus

$$x \in b \subset b \cup c \subset \bigcup \zeta(U),$$

therefore $\bigcup U \subseteq \bigcup \zeta(U)$ and the equality holds. This concludes the proof. □

Lemma 9.1.8. *Let $U \subseteq V_{n,m}$, $G = G[U]$, $G' = G[\zeta(U)]$ and $\mathcal{U}$ be the set of connected components of the graph G. Then the following conditions hold:*

1. *For any connected component V of G', there exists a set $\mathcal{O} \subseteq \mathcal{U}$ such that*

$$V = \zeta(\bigcup \mathcal{O}) = \bigcup_{Z \in \mathcal{O}} \zeta(Z).$$

2. *If V, V' are two different connected components of G' and $\mathcal{O}, \mathcal{O}' \subseteq \mathcal{U}$ are the sets which fulfill property 1 for V and V' respectively, then $\mathcal{O} \cap \mathcal{O}' = \varnothing$.*

Proof. Part 2 is clearly true, so we only need to prove part 1. Define the graph $\mathcal{H} = (V(\mathcal{H}), E(\mathcal{H}))$ as follows:

$V(\mathcal{H}) = \mathcal{U}$;

Two vertices Z, Z' form an edge in $E(\mathcal{H})$ if and only if there exist $b, c \in Z$ and $d, e \in Z'$ such that

- $|b \cap c| = m - 1$ and $|d \cap e| = m - 1$
- $\big|(b \cup c) \cap (d \cup e)\big| \geqslant m$.

Roughly speaking, $\mathcal{H}$ describes for two connected components Z, Z' of G, whether $\zeta(Z), \zeta(Z')$ lie in the same connected component of G' or not. Let V be a connected component of G', if $b \in V$, then $b = b_1 \cup b_2$, where b_1, b_2 are in some connected component $Z_b \subseteq U$ of G. In $\mathcal{H}$, there exists a component $\mathcal{O}$ such that $Z_b \in \mathcal{O}$. By Lemma 9.1.7, $G[\zeta(Z_b)]$ is connected and $b \in \zeta(Z_b) \cap V$, so it is clear that $\zeta(Z_b) \subseteq V$. Suppose that $Z' \in \mathcal{O}$ with $Z' \neq Z_b$ and these components form an edge in $\mathcal{H}$. This means that $G[\zeta(Z_b)]$ and $G[\zeta(Z')]$ are joined by at least one edge in G', thus every vertex of $\zeta(Z')$ is connected with every vertex of $\zeta(Z_b)$, and as $\zeta(Z_b) \subseteq V$, $\zeta(Z_b) \cup \zeta(Z') \subseteq V$. We can continue this process with every element of the set $\mathcal{O} - \{Z_b, Z'\}$ to show that

$$\zeta(\bigcup \mathcal{O}) = \bigcup_{Z \in \mathcal{O}} \zeta(Z) \subseteq V.$$

(The first equality comes from part (ii) of Lemma 9.1.4). We prove the other inclusion, if $V = \{b\}$, we are done. Otherwise, let $c \in V - \{b\}$, if $c \in \zeta(Z_b)$, then $c \in \zeta(\bigcup \mathcal{O})$. In case that $c \notin \zeta(Z_b)$, there must exists some connected component Z_c of G such that $c \in \zeta(Z_c)$ and $Z_c \neq Z_b$. In G', we can find a path $b = v_1, \ldots, v_q = c$ where $v_i \in V$ for $i = 1, \ldots, q$. The set $\mathcal{Q} \subset \mathcal{U}$ defined as

$$\mathcal{Q} = \{X \in \mathcal{U} \mid (\exists j)(1 \leqslant j \leqslant q \text{ and } v_j \in \zeta(X))\},$$

can be seen to have the property that every pair of vertices $X, X' \in \mathcal{Q}$ are connected by a path in $\mathcal{H}$. Since $Z_b, Z_c \in \mathcal{Q}$, they are connected in $\mathcal{H}$ and as $Z_b \in \mathcal{O}$, then $Z_c \in \mathcal{O}$, so that $c \in \zeta(\bigcup \mathcal{O})$. Therefore $V \subseteq \zeta(\bigcup \mathcal{O})$ and the equality $V = \bigcup_{Z \in \mathcal{O}} \zeta(Z)$ holds. This proves part 1 and finishes the proof. □

Lemma 9.1.9. *Let $U \subseteq V_{n,m}$ and $\mathcal{U}$ be the set of connected components of $G[U]$. Then for all $s \in \{0, \ldots, n-m\}$ and $G^s = G[\zeta^s(U)]$, the following conditions are satisfied.*

H1 *For every connected component V of the graph G^s, there exists a set $\mathcal{O} \subseteq \mathcal{U}$ such that*

$$|\bigcup V| \leqslant m - 1 + \sum_{Z \in \mathcal{O}} |Z|. \tag{9.6}$$

H2 *If V, V' are two different connected components of G^s and $\mathcal{O}, \mathcal{O}' \subseteq \mathcal{U}$ are the sets which make true the inequality given in (9.6) for V and V' respectively, then $\mathcal{O} \cap \mathcal{O}' = \varnothing$.*

Proof. We prove the lemma by using induction on s. For the base case $s = 0$, $G^0 = G[U]$, we use Lemma 9.1.4 and we are done. Suppose that for $0 \leqslant s < n - m$, **H1** and **H2** are true. We prove the case $s + 1$, let W be a connected component of the graph G^{s+1}. By part 1 of Lemma 9.1.8, we know that there is a unique set $\mathcal{Q}$ of connected components of G^s such that

$$W = \bigcup_{Q \in \mathcal{Q}} \zeta(Q)$$

and because $\zeta(Q) \neq \varnothing$, $|Q| > 1$, so that by Lemma 9.1.7, $\bigcup \zeta(Q) = \bigcup Q$ for all $Q \in \mathcal{Q}$, thus it is true that

$$\begin{aligned} \bigcup W &= \bigcup (\bigcup_{Q \in \mathcal{Q}} \zeta(Q)) \\ &= \bigcup_{Q \in \mathcal{Q}} (\bigcup \zeta(Q)) \\ &= \bigcup_{Q \in \mathcal{Q}} (\bigcup Q). \end{aligned}$$

By the induction hypothesis, $|\bigcup Q| \leqslant m - 1 + \sum_{Z \in \mathcal{O}_Q} |Z|$, where $\mathcal{O}_Q \subseteq \mathcal{U}$ for all $Q \in \mathcal{Q}$ and when $Q \neq R$, $\mathcal{O}_Q \cap \mathcal{O}_R = \varnothing$. With a simple induction on $|\mathcal{Q}|$, it is rather forward to show that

$$|\bigcup W| \leqslant m - 1 + \sum_{Z \in \mathcal{O}} |Z|,$$

where $\mathcal{O} = \bigcup_{Q \in \mathcal{Q}} \mathcal{O}_Q$, thus property **H1** is fulfilled. We now show that condition **H2** holds. If W' is another connected component of G^{s+1}, then applying the above procedure to W' yields $W' = \bigcup_{S \in \mathcal{S}} \zeta(S)$, $\bigcup W' = \bigcup_{S \in \mathcal{S}} (\bigcup S)$ and $|\bigcup W'| \leqslant m - 1 + \sum_{X \in \mathcal{O}'} |X|$, with $\mathcal{O}' = \bigcup_{S \in \mathcal{S}} \mathcal{O}_S$. By 2 of Lemma 9.1.8, $\mathcal{Q} \cap \mathcal{S} = \varnothing$, so that if $Q \in \mathcal{Q}$ and $S \in \mathcal{S}$, then $Q \neq S$ and the induction hypothesis tell us that $\mathcal{O}_Q \cap \mathcal{O}_S = \varnothing$, thus

$$\begin{aligned} \mathcal{O} \cap \mathcal{O}' &= \Big(\bigcup_{Q \in \mathcal{Q}} \mathcal{O}_Q\Big) \cap \Big(\bigcup_{S \in \mathcal{S}} \mathcal{O}_S\Big) \\ &= \bigcup_{(Q,S) \in \mathcal{Q} \times \mathcal{S}} (\mathcal{O}_Q \cap \mathcal{O}_S) \\ &= \varnothing, \end{aligned}$$

and property **H2** is satisfied, so that by induction we obtain the result. □

We can now prove the main result of the combinatorial part of the proof of the lower bound.

Theorem 9.1.10. *Let $U \subset V_{n,m}$ such that $|U| \leqslant n - m$. Then $\zeta^{n-m}(U) = \varnothing$.*

Proof. For a contradiction, suppose that $\zeta^{n-m}(U) \neq \varnothing$, then $G\left[\zeta^{n-m}(U)\right]$ is the graph $J_{n,n}$ and contains the unique connected component $C = \{\overline{n}\}$, thus by Lemma 9.1.9, for some set $\mathcal{O}$ of connected components of $G\left[U\right]$,

$$n = \left|\bigcup C\right| \leqslant m - 1 + \sum_{Z \in \mathcal{O}} |Z| \leqslant m - 1 + |U|,$$

and we conclude that $|U| \geqslant n - m + 1$, a contradiction. So that $\zeta^{n-m}(U) \neq \varnothing$ is impossible. Therefore $\zeta^{n-m}(U)$ has no elements. □

9.2 The combinatorics of the lower bound

In this section, we give a general overview of the lower bound proof, by using the consensus protocol of Chapter 7 to present two examples that illustrate the combinatorial interactions between the sets of processes that invoke safe-consensus shared objects (represented by boxes) and the sets of processes that cannot distinguish between states of a path of connected reachable states in the protocol. These examples will help us to understand the intuitive details of the lower bound proof.

9.2.1 The degree of indistinguishability of a path and safe-consensus invocations

Roughly, a typical consensus impossibility proof shows that a protocol $\mathcal{A}$ cannot solve consensus because, given one execution of $\mathcal{A}$ in which processes decide a consensus value v, and a second execution of $\mathcal{A}$ where the consensus output is v' ($v \neq v'$), the global states of these executions can be connected with paths of connected states, implying that some processes decide distinct output values [FLP85, LAA87, CR12], violating the Agreement requirement of consensus. Any protocol that is able to solve consensus, must prevent the existence of such paths of connected states.

For the case of our lower bound proof, the main circumstance that will prevent $\mathcal{A}$ from solving consensus is that for some $m \in \{2, \ldots, n\}$, it is true that $\nu_{\mathcal{A}}(n,m) \leqslant n - m$, i.e., at most $n - m$ subsets of processes of size m can invoke safe-consensus shared objects in $\mathcal{A}$.

We introduce two examples using the consensus protocol of Chapter 7. The first example explains how the protocol works its way to separate connected global states so that the processes can reach consensus; while in the second example we modify the protocol to obtain a bad protocol which we use to explain how the lower bound proof finds a path of connected states which cannot be separated by the modified protocol and because of this, the processes fail to make a consensus.

Example: Reaching consensus.

We use the consensus protocol $\mathcal{A}$ from Chapter 7 to illustrate how a working protocol is able to destroy a path of connected states using the invocations of the safe-consensus objects. Suppose that $n = 4$ and let I_0, I_1 be the two initial states of $\mathcal{A}$ such that, all processes have as input values 0's in I_0 and 1's in I_1. Using Lemma 5.4.3, we can show easily that I_0, I_1 are connected with a path of initial states

$$\mathfrak{p}_0 \colon I_0 \overset{\{1,2,3\}}{\sim} J_1 \overset{\{1,2,4\}}{\sim} J_2 \overset{\{1,3,4\}}{\sim} J_3 \overset{\{2,3,4\}}{\sim} I_1, \qquad \boxed{9.7}$$

and now assume that the processes begin executing $\mathcal{A}$ (part (a) of Figure 9.3). After the protocol has been executed for three rounds, the processes have invoked three safe-consensus shared objects represented by the 2-boxes $\{1,2\}, \{2,3\}, \{3,4\} \in \Gamma_{\mathcal{A}}(4,2)$. With the invocations of these safe-consensus objects, the processes can "weaken" the path $\mathfrak{p}_0$ in part (b) of Figure 9.3. Notice how the sets $b_1 = \{3,4\}, X_1 = \{1,2,3\}$ and $X_2 = \{1,2,4\}$ satisfy the following combinatorial property

$$|b_1 \cap X_1| = 1 \text{ and } |b_1 \cap X_2| = 1. \qquad \boxed{9.8}$$

This important property allows the safe-consensus object invoked by p_3 and p_4 to be able to weaken the path $\mathfrak{p}_0$. It can be checked that if b_1 does not fulfill $\boxed{9.8}$, then we can find an execution of $\mathcal{A}$ in which the shared object represented by b_1 is "useless", it does not help us to weaken $\mathfrak{p}_0$, because we can use the Safe-Validity property of the safe-consensus task to find executions of $\mathcal{A}$ in which the value returned to the processes by the shared object represented by b_1 is the same in all these executions, thus this value cannot help the processes to distinguish global states.

It can be seen that $b_2 = \{2,3\}, X_2$ and $X_3 = \{1,3,4\}$ satisfy similar statements and the same is true for $b_3 = \{1,2\}, X_3$ and $X_4 = \{2,3,4\}$. In summary, after three rounds of $\mathcal{A}$, using the safe-consensus objects represented by b_1, b_2 and b_3, the processes are able to weaken the path $\mathfrak{p}_0$, and the result is a new path $\mathfrak{p}_1$ (Figure 9.3 (c)), which is a path connecting the states I_0^3, I_1^3, which are successor states of I_0 and I_1. A fundamental property of $\mathfrak{p}_1$ is that $\deg \mathfrak{p}_1 = 2$, being $X_5 = \{1,2\}, X_6 = \{1,4\}$ and $X_7 = \{3,4\}$ the only elements in $\mathsf{iSets}(\mathfrak{p}_1)$ with cardinality 2, while in $\mathfrak{p}_0$, all the sets X_l $(l = 1,2,3,4)$ have size three, thus $\deg \mathfrak{p}_0 = 3$. It is in this sense that the new path $\mathfrak{p}_1$ is weaker that $\mathfrak{p}_0$ and there is no way to find a path of successor states of I_0, I_1 which does not satisfy this property. The set of 2-boxes which satisfy property $\boxed{9.8}$ for the path $\mathfrak{p}_0$ is very important, let

$$\beta_{\mathcal{A}}(\mathfrak{p}_0; 2) = \{b \in \Gamma_{\mathcal{A}}(4,2) \mid (\exists X, Y \in \mathsf{iSets}(\mathfrak{p}_0))(|X \cap b| = 1 \wedge |Y \cap b| = 1)\},$$

thus, $\beta_{\mathcal{A}}(\mathfrak{p}_0; 2) = \{\{1,2\}, \{2,3\}, \{3,4\}\}$ and this is the set of boxes which can weaken the path $\mathfrak{p}_0$, in the sense described above.

Now back to the execution of $\mathcal{A}$. If the processes continue to execute the protocol, they have to work their way through the path $\mathfrak{p}_1$ (part (c) of Figure 9.3). The shared objects that they will invoke in the next two rounds are represented by $b_4 = \{2,3,4\}$ and $b_5 = \{1,2,3\}$ $(b_4, b_5 \in \Gamma_{\mathcal{A}}(4,3))$ in part (d) of Figure 9.3. A simple analysis will show us that b_4 satisfies Equation $\boxed{9.8}$ with the sets X_5, X_6 and the same is true for b_5, X_6 and X_7. Again, it can be checked that if these conditions are not fulfilled, then the path $\mathfrak{p}_1$ cannot be weakened by the shared objects represented by the boxes b_4, b_5 and then the processes will not make any progress to reach a consensus. The result of applying the safe-consensus objects represented by b_4, b_5 is the path $\mathfrak{p}_2$ (Figure 9.3 (e)) with $\deg \mathfrak{p}_2 = 1$. As

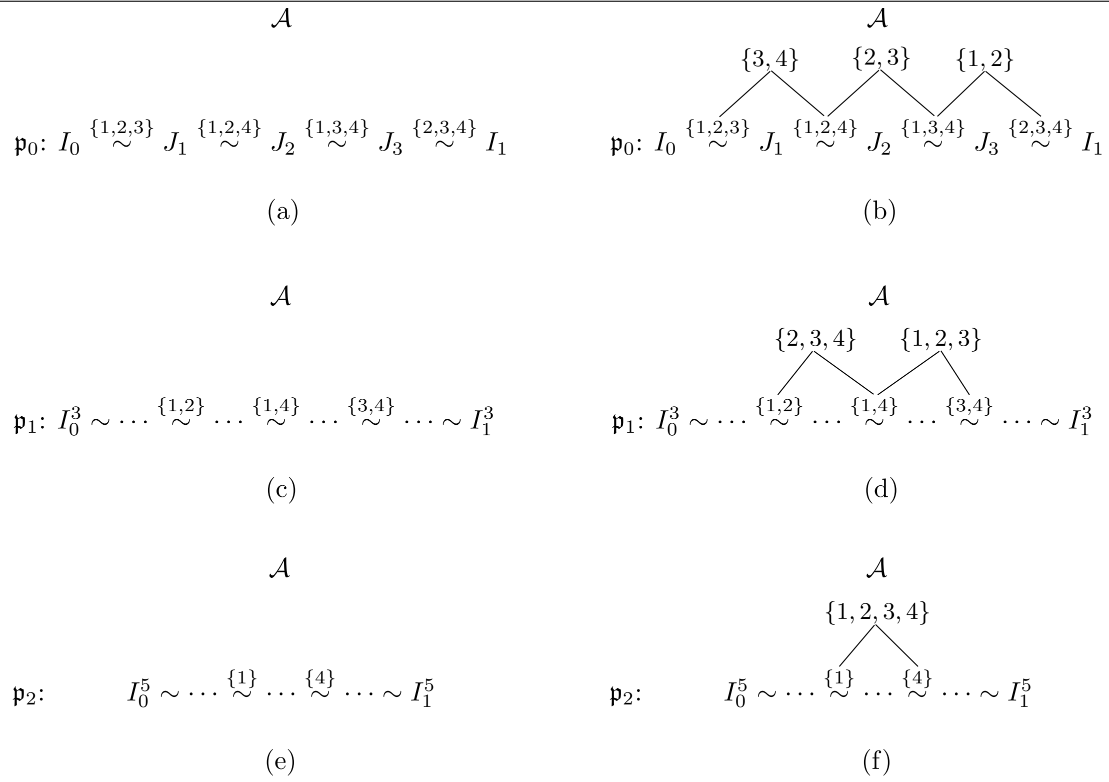

$\mathcal{A}$
$\mathfrak{p}_0$: $I_0 \overset{\{1,2,3\}}{\sim} J_1 \overset{\{1,2,4\}}{\sim} J_2 \overset{\{1,3,4\}}{\sim} J_3 \overset{\{2,3,4\}}{\sim} I_1$
(a)
$\{3,4\}$ $\{2,3\}$ $\{1,2\}$
(b)
$\mathfrak{p}_1$: $I_0^3 \sim \cdots \overset{\{1,2\}}{\sim} \cdots \overset{\{1,4\}}{\sim} \cdots \overset{\{3,4\}}{\sim} \cdots \sim I_1^3$
(c)
$\{2,3,4\}$ $\{1,2,3\}$
(d)
$\mathfrak{p}_2$: $I_0^5 \sim \cdots \overset{\{1\}}{\sim} \cdots \overset{\{4\}}{\sim} \cdots \sim I_1^5$
(e)
$\{1,2,3,4\}$
(f)

Figure 9.3: The combinatorial interactions between paths of connected states of an iterated protocol $\mathcal{A}$ and the safe-consensus objects invoked by the processes.

with the path $\mathfrak{p}_0$, the set of boxes which can weaken $\mathfrak{p}_1$ is given by $\beta_{\mathcal{A}}(\mathfrak{p}_1;3) = \{\{1,2,3\},\{2,3,4\}\}$ (this set is defined in a similar way as $\beta_{\mathcal{A}}(\mathfrak{p}_0;2)$ is defined) and notice that $\beta_{\mathcal{A}}(\mathfrak{p}_0;2)$ and $\beta_{\mathcal{A}}(\mathfrak{p}_1;3)$ are subsets of vertices of $J_{4,2}$ and $J_{4,3}$ respectively and they satisfy the relation

$$\beta_{\mathcal{A}}(\mathfrak{p}_1;3) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{p}_0;2)).$$

Following the same argument, in the last round of $\mathcal{A}$, all the processes invoke a safe-consensus object represented by the set $b_6 = \{1,2,3,4\} \in \Gamma_{\mathcal{A}}(4,4)$, as shown in part (f) of Figure 9.3. We can prove easily that

$$|b_6 \cap X_8| = |b_6 \cap X_9| = 1, \tag{9.9}$$

where $X_8 = \{2\}$ and $X_9 = \{4\}$. Thus $\beta_{\mathcal{A}}(\mathfrak{p}_2;4) = \{\{1,2,3,4\}\}$ and we can also prove that b_6 is the unique set that can satisfy (9.9). As a result of this action, there cannot be a new path connecting successor states of I_0 and I_1. Thus the set of paths $\{\mathfrak{p}_0,\mathfrak{p}_1,\mathfrak{p}_2\}$ has been destroyed by the processes and they finally take a decision on a unique consensus value. Finally, we have the relations

$$\varnothing \neq \beta_{\mathcal{A}}(\mathfrak{p}_2;4) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{p}_1;3)) \subseteq \zeta^2(\beta_{\mathcal{A}}(\mathfrak{p}_0;2)). \tag{9.10}$$

A general form of Equations (9.8) and (9.10) play an important role in the lower bound proof. We will prove that given any path $\mathfrak{s}$ of connected states of a protocol, we can build a sequence of paths of successor states such that Equation (9.10) is satisfied. Also, the proof of the lower bound shows that the general form of the condition $\beta_{\mathcal{A}}(\mathfrak{p}_2;4) \neq \varnothing$ of the previous example must be satisfied for any iterated protocol that solves consensus using safe-consensus objects.

We remark that as we have proven in Chapter 7 that the protocol $\mathcal{A}$ is a correct consensus protocol, it will do the same thing to any path of initial states that we propose, using only the invocations of safe-consensus objects that the processes do.

Example: The lower bound proof.

To illustrate the lower bound proof,we now present a modified protocol, which is unable to solve consensus and explain why. Suppose that we modify the protocol of Chapter 7 in such a way that the only invocations of safe-consensus objects done by two processes are the 2-boxes $c_1 = \{3,4\}$ and $c_2 = \{1,2\}$. We obtain a new protocol $\mathcal{A}'$ with the property that

$$\nu_{\mathcal{A}'}(4,2) = 2 = 4 - 2.$$

Assume now that the processes begin the execution of $\mathcal{A}'$ and take the path $\mathfrak{p}_0$ of Equation $\boxed{9.7}$. When the processes try to weaken the path $\mathfrak{p}_0$ (Figure 9.4 (b′)), with the safe-consensus objects represented by the boxes $c_1, c_2 \in \Gamma_{\mathcal{A}'}(4,2)$, we can prove that there are executions of $\mathcal{A}$ in which we can obtain successor states I'_0, I'_1 of I_0 and I_1 respectively, connected with a path $\mathfrak{p}'_1$ which is shown in part (b′) of Figure 9.4. The most important property of $\mathfrak{p}'_1$ is the following: The sets $Y_1 = \{1,2\}$ and $Y_2 = \{3,4\}$ are the smallest sets of ids of processes that cannot distinguish between some pairs of states of $\mathfrak{p}'_1$, thus $\deg \mathfrak{p}'_1 = 2$ and for the set $\beta_{\mathcal{A}'}(\mathfrak{p}_0;2)$, we have that $\beta_{\mathcal{A}'}(\mathfrak{p}_0;2) = \{\{1,2\},\{3,4\}\}$ and these are the only boxes which can weaken the path $\mathfrak{p}_0$.

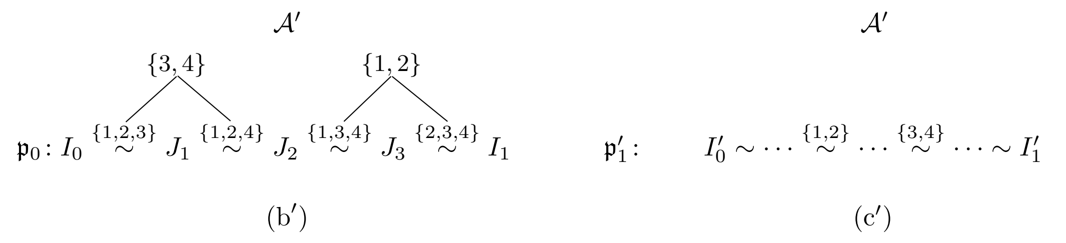

$\mathcal{A}'$
$\{3,4\}$
$\{1,2\}$
$\mathfrak{p}_0 : I_0 \overset{\{1,2,3\}}{\sim} J_1 \overset{\{1,2,4\}}{\sim} J_2 \overset{\{1,3,4\}}{\sim} J_3 \overset{\{2,3,4\}}{\sim} I_1$
(b′)
$\mathcal{A}'$
$\mathfrak{p}'_1 : \quad I'_0 \sim \cdots \overset{\{1,2\}}{\sim} \cdots \overset{\{3,4\}}{\sim} \cdots \sim I'_1$
(c′)

Figure 9.4: The obstruction $\{\mathfrak{p}_0, \mathfrak{p}'_1, \ldots\}$ to reach consensus for the case $\nu_{\mathcal{A}'}(4,2) \leqslant 2$.

If now we try to mimic the behavior of the previous example with the execution of the protocol $\mathcal{A}'$, then to weaken the connectivity of $\mathfrak{p}'_1$, some processes must invoke a safe-consensus object, represented by a box c, such that

$$|c \cap Y_1| = 1 \text{ and } |c \cap Y_2| = 1, \qquad \boxed{9.11}$$

or, in other words, we must find a box $c \in \beta_{\mathcal{A}'}(\mathfrak{p}'_1;3)$. But it can be proven that, as in the previous example

$$\beta_{\mathcal{A}'}(\mathfrak{p}'_1;3) \subseteq \zeta(\beta_{\mathcal{A}'}(\mathfrak{p}_0;2))$$

But using Definition 9.1.2, it is easy to check that $\zeta(\beta_{\mathcal{A}'}(\mathfrak{p}_0;2)) = \varnothing$, thus there is no set $c \subseteq \overline{n}$ that can satisfy Equation $\boxed{9.11}$ with Y_1 and Y_2. This fact can be used to find executions of $\mathcal{A}$ in which the output value of all the safe-consensus shared objects represented by all the boxes of size $m \geqslant 3$ is fixed for all these executions, thus these shared objects will not help the processes to distinguish between some specific global states. As a result of this, we will be able to build a new path $\mathfrak{p}_2'$ with the same properties as $\mathfrak{p}_1'$ and such that

$$\deg \mathfrak{p}_2' \geqslant \deg \mathfrak{p}_1'.$$

Moreover, we can find a path $\mathfrak{p}_u'$ ($u \geqslant 2$) for any subsequent round of $\mathcal{A}$ and such that $\deg \mathfrak{p}_u' \geqslant \deg \mathfrak{p}_1'$. This fact can be used to build a bivalency argument to show that $\mathcal{A}'$ cannot solve consensus. We can say that the infinite sequence of paths $\mathfrak{p}_0, \mathfrak{p}_1', \dots, \mathfrak{p}_u', \dots$ is an "obstruction" for $\mathcal{A}'$ to solve the consensus task.

The main circumstance that prevents the protocol $\mathcal{A}'$ from solving consensus, is that $\Gamma_{\mathcal{A}'}(4,2) = \beta_{\mathcal{A}'}(\mathfrak{p}_0; 2) = \{\{1,2\},\{3,4\}\}$, which implies, for this example, that $\zeta(\beta_{\mathcal{A}'}(\mathfrak{p}_0; 2)) = \varnothing$ and this will imply that

$$\beta_{\mathcal{A}'}(\mathfrak{p}_l; 4) = \varnothing, \quad \mathfrak{p}_l \in \{\mathfrak{p}_0, \mathfrak{p}_1', \dots, \mathfrak{p}_u', \dots\} \text{ and } l \geqslant 2$$

In the general setting, we will use the fact that when for some $m_0 \in \overline{n}$, $\nu_{\mathcal{A}'}(n, m_0) = |\Gamma_{\mathcal{A}'}(n, m_0)| \leqslant n - m_0$, then $\zeta^{n-m_0}(\Gamma_{\mathcal{A}'}(n, m_0)) = \varnothing$ (Theorem 9.1.10) and this will allow us to find executions of $\mathcal{A}'$ in which we can find an infinite sequence of paths of connected states,where these paths connect successor states of some initial states with different valencies.

9.3 The formal results

We have gathered all the elements needed to prove our lower bound result. As we have said before, we will show that if $\mathcal{A}$ is an iterated protocol for n-consensus, such that for some m, $\nu_{\mathcal{A}}(n,m) \leqslant n - m$, then there exists an execution of $\mathcal{A}$ in which two processes decide distinct output values. Before proving our lower bound, we need a series of technical results that describe the global structure of iterated protocols with safe-consensus shared objects. One of these results formalizes all the ideas described in the previous section with the two given examples of an executing protocol and paths of connected reachable states.

9.3.1 Results for $3 \leqslant m \leqslant n$

Before proving the lower bound, for technical reasons, we need to divide our results in two cases. The first case covers the inequality $\nu_{\mathcal{A}}(n,m) \leqslant n - m$ for any iterated protocol $\mathcal{A}$ with safe-consensus and $m \in \{3, \dots, n\}$. The main result in this case is Theorem 9.3.3. For a given path $\mathfrak{s}$ of connected states of an iterated protocol $\mathcal{A}$, consider the set

$$\beta_{\mathcal{A}}(\mathfrak{s}) = \{b \in \Gamma_{\mathcal{A}}(n) \mid (\exists X, Y \in \mathsf{iSets}(\mathfrak{s}))(|X \cap b| = 1 \land |Y \cap b| = 1)\}.$$

The set $\beta_{\mathcal{A}}(\mathfrak{s})$ contains all boxes of the protocol $\mathcal{A}$ which can potentially weaken the path $\mathfrak{s}$, in the sense of the two examples of Section 9.2. For example, given the consensus protocol of Chapter 7 for four processes and the path $\mathfrak{p}_0$ of Figure 9.3, we have that $\beta_{\mathcal{A}}(\mathfrak{p}_0) = \{\{3,4\},\{2,3\},\{1,2\}\}$. If $m \in \overline{n}$, let

$$\beta_{\mathcal{A}}(\mathfrak{s}; m) = \beta_{\mathcal{A}}(\mathfrak{s}) \cap \Gamma_{\mathcal{A}}(n,m).$$

Notice that $\beta_{\mathcal{A}}(\mathfrak{s}; m) \subseteq V_{n,m}$ (see Section 9.1).

The next lemma is the basic tool in proving Theorem 9.3.3, we start with a path $\mathfrak{s}$ of connected states with various useful properties, related to the sets $\Gamma_{\mathcal{A}}(n,m)$ (specifically, the complements of some boxes of $\mathcal{A}$ are elements of the set $\mathsf{iSets}(\mathfrak{s})$) and as result, a new path $\mathfrak{q}$ is built (in the next iteration of $\mathcal{A}$) such that $\mathfrak{q}$ extends the properties of $\mathfrak{s}$.

Lemma 9.3.1. *Let $n \geqslant 3$ and $1 \leqslant v \leqslant s \leqslant n-2$ be fixed. Suppose that $\mathcal{A}$ is an iterated protocol with safe-consensus objects and there exists a path*

$$\mathfrak{s}\colon S_0 \overset{X_1}{\sim} \cdots \overset{X_q}{\sim} S_q \quad (q \geqslant 1),$$

of connected states of round $r \geqslant 0$ such that

Λ_1 $\deg \mathfrak{s} \geqslant v$.

Λ_2 *For every X_i with $v \leqslant |X_i| < s$, there exists $b_i \in \Gamma_{\mathcal{A}}(n, n-|X_i|)$ such that $X_i = \overline{n} - b_i$*

Λ_3 $\overline{n} \notin \beta(\mathfrak{s})$.

Then in round $r+1$ there exists a path

$$\mathfrak{q}\colon Q_0 \sim \cdots \sim Q_u,$$

of connected successor states of all the S_j, such that the following statements hold:

Ψ_1 *If $\beta_{\mathcal{A}}(\mathfrak{s}; n-v+1) = \varnothing$, then* $\deg \mathfrak{q} \geqslant v$.

Ψ_2 *If $\beta_{\mathcal{A}}(\mathfrak{s}; n-v+1)$ is not empty, then* $\deg \mathfrak{q} \geqslant v-1$.

Ψ_3 *For every $Z \in \mathsf{iSets}(\mathfrak{q})$ with $|Z| = v-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{s})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{s}; n-v+1)$ such that $Z = \overline{n} - b = X \cap X'$ and $b = c_1 \cup c_2$, $c_k \in \Gamma_{\mathcal{A}}(n, n-v)$.*

Ψ_4 *For every $Z \in \mathsf{iSets}(\mathfrak{q})$ with $v \leqslant |Z| < s$, there exists a unique $b \in \Gamma_{\mathcal{A}}(n, n-|Z|)$ such that $Z = \overline{n} - b$.*

Ψ_5 *$\beta_{\mathcal{A}}(\mathfrak{q}; n-l+1) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{s}; n-l))$ for $v \leqslant l < s$.*

Proof. In order to find the path $\mathfrak{q}$ that has the properties Ψ_1-Ψ_5, first we need to define a set of states of round $r+1$ of $\mathcal{A}$, called $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$, which we will use to construct the path $\mathfrak{q}$. The key ingredient of $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ is the safe-consensus values of the boxes used in each member of $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$. Define the set of states $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ as

$$\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s}) = \{R \mid R = S \cdot \xi(X) \wedge (S, X) \in \mathsf{States}(\mathfrak{s}) \times \mathsf{iSets}(\mathfrak{s})\}.$$

Each element of $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ is a state in round $r+1$ of $\mathcal{A}$ which is obtained when all the processes with ids in some set $X \in \mathsf{iSets}(\mathfrak{s})$ execute concurrently the operations of $\mathcal{A}$, followed by all processes with ids in $\overline{n} - X$. The safe-consensus values of each box for the states of $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ in round $r+1$ are defined using the following rules: Let $b \in \Gamma_{\mathcal{A}}(n)$ be any box such that $b \notin \beta(\mathfrak{s})$.

- If $|X \cap b| \neq 1$ for every $X \in \mathsf{iSets}(\mathfrak{s})$, then we claim that we can choose any element $j \in \mathbb{N}$ such that j is the safe-consensus value of b in every state $R \in \Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ such that $b \in \mathbf{Inv}(R)$.
- If there exists exactly one set $X \in \mathsf{iSets}(\mathfrak{s})$ such that $X \cap b = \{x\}$, then b has x as its safe-consensus value in every state $Q \in \Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ with $b \in \mathbf{Inv}(Q)$.

We establish rules to define the safe-consensus values of every element of the set $\beta_{\mathcal{A}}(\mathfrak{s})$, using the order on the set $\mathsf{iSets}(\mathfrak{s})$ induced by the path $\mathfrak{s}$, when we traverse $\mathfrak{s}$ from S_0 to S_q. So that $\mathsf{iSets}(\mathfrak{s})$ is ordered as

$$X_1, \ldots, X_q. \tag{9.12}$$

For each $b \in \beta_{\mathcal{A}}(\mathfrak{s})$, suppose that $X_i, X_{i+z_1}, \ldots, X_{i+z_k}$, $(1 \leqslant i < i+z_1 < i+z_2 < \cdots < i+z_k \leqslant q; z_j > 0$ and $k \geqslant 1)$ is the (ordered) subset of $\mathsf{iSets}(\mathfrak{s})$ of all $X \in \mathsf{iSets}(\mathfrak{s})$ with the property $|X \cap b| = 1$. Take the set X_i. If $x_i \in X_i \cap b$, then x_i is the safe-consensus value of b for all the elements of $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ of the form $P_j = S \cdot \xi(X_j)$ where $b \in \mathbf{Inv}(P_j)$ and $1 \leqslant j \leqslant i+z_1-1$. Next, in all states $R_u \in \Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ such that $R_u = S \cdot \xi(X_u)$ with $b \in \mathbf{Inv}(R_u)$ and $i+z_1 \leqslant u \leqslant i+z_2-1$, b has safe-consensus value equal to $x_{i+z_1} \in X_{i+z_1} \cap b$. In general, in all states $T_v \in \Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ of the form $T_v = S \cdot \xi(X_v)$ where $b \in \mathbf{Inv}(T_v)$ and $i+z_l \leqslant v \leqslant i+z_{l+1}-1$ and $1 \leqslant l \leqslant k-1$, b has safe-consensus value equal to $x_{i+z_l} \in X_{i+z_l} \cap b$. Finally, in each state $L_w = S \cdot \xi(X_w) \in \Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ with $b \in \mathbf{Inv}(L_w)$ and $i+z_k \leqslant w \leqslant q$, b has safe-consensus value equal to $x_{i+z_k} \in X_{i+z_k} \cap b$.

Using the Safe-Validity property of the safe-consensus task, it is an easy (but long) routine task to show that we can find states of $\mathcal{A}$ in round $r+1$ which have the form of the states given in $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ and with the desired safe-consensus values for every box.

We are ready to build the path $\mathfrak{q}$ of the Lemma[1]. We use induction on $q \geqslant 1$. For the base case when $q = 1$, we have that $\mathfrak{s}\colon S_0 \overset{X_1}{\sim} S_1$, with $|X_1| \geqslant v$. Here we easily build the path $S_0 \cdot \xi(X_1) \overset{X_1}{\sim} S_1 \cdot \xi(X_1)$ which clearly satisfies properties Ψ_1-Ψ_5. Assume that for $1 \leqslant w < q$ we have a path

$$\mathfrak{q}'\colon Q_0 \sim \cdots \sim Q_l \quad (l \geqslant 1),$$

Satisfying conditions Ψ_1-Ψ_5 and such that $Q_0 = S_0 \cdot \xi(X_1), Q_l = S_w \cdot \xi(X_w)$. We now connect the state Q_l with $Q = S_{w+1} \cdot \xi(X_{w+1})$. By Theorem 8.2.5, there is a C-regular path

$$\mathfrak{v}\colon Q_l \sim \cdots \sim Q'$$

such that $Q' = S_w \cdot \xi(X_{w+1})$ and $\mathfrak{v}, Q_l$ and Q' satisfy conditions (A)-(B) of that theorem. Clearly $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q') \subseteq \beta_{\mathcal{A}}(\mathfrak{s})$, because the only boxes used by $\mathcal{A}$ in the states Q_l and Q' with different safe-consensus value, are the boxes which intersect two different sets $X, X' \in \mathsf{iSets}(\mathfrak{s})$ in one element. Joining the path $\mathfrak{q}'$ with $\mathfrak{v}$, followed by the small path $\mathfrak{t}\colon Q' \overset{X_{w+1}}{\sim} Q$ (of degree at least v), we obtain a new path $\mathfrak{q}\colon Q_0 \sim \cdots \sim Q$. We now need to show that $\mathfrak{q}$ satisfies properties Ψ_1-Ψ_5.

Ψ_1 Notice that every box in $\beta(\mathfrak{s})$ has size no bigger that $n-v+1$. Suppose that $\beta_{\mathcal{A}}(\mathfrak{s}; n-v+1) = \varnothing$. In particular, this implies that $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q') \cap \Gamma_{\mathcal{A}}(n, n-v+1) = \varnothing$. If it happens that $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q')$ is void, then as condition (A) of Theorem 8.2.5 holds for $\mathfrak{v}$, $\deg \mathfrak{v} \geqslant v$. But if $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q') \neq \varnothing$ we have that $|b| \leqslant n-v$ for all $b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q')$, thus by part 1. of property (B) of Theorem 8.2.5,

$$\deg \mathfrak{v} \geqslant \min\{n - |b|\}_{b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q')} \geqslant v.$$

By the induction hypothesis, $\deg \mathfrak{q}' \geqslant v$ and also $\deg \mathfrak{t} \geqslant v$, so that $\deg \mathfrak{q} \geqslant v$.

[1]Notice that the order given to $\Phi^{r+1}_{\mathcal{A}}(\mathfrak{s})$ in equation 9.12 is the precise order in which the states of this set will appear in the path $\mathfrak{q}$.

Ψ_2 If $\beta_{\mathcal{A}}(\mathfrak{s}; n-v+1) \neq \varnothing$, then either $\deg \mathfrak{q}' \geqslant v-1$ (by the induction hypothesis) or $\deg \mathfrak{v} \geqslant v-1$. This last assertion is true, because by (B) of Theorem 8.2.5, we have that $\deg \mathfrak{v} \geqslant \min\{n-|b|\}_{b\in\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l,Q')} \geqslant v-1$. Therefore $\deg \mathfrak{q} \geqslant v-1$.

Ψ_3 We remark first that $\mathsf{iSets}(\mathfrak{q}) = \mathsf{iSets}(\mathfrak{q}') \cup \mathsf{iSets}(\mathfrak{v}) \cup \mathsf{iSets}(\mathfrak{t})$. If we are given $Z \in \mathsf{iSets}(\mathfrak{q})$ such that $|Z| = v-1$, then $Z \in \mathsf{iSets}(\mathfrak{q}')$ or $Z \in \mathsf{iSets}(\mathfrak{v})$. When $Z \in \mathsf{iSets}(\mathfrak{q}')$, we use our induction hypothesis to show that $Z = Y \cap Y' = \overline{n} - d$ for some $Y, Y' \in \mathsf{iSets}(\mathfrak{s})$ and unique $d \in \beta_{\mathcal{A}}(\mathfrak{s}; n-v+1)$ such that $d = d' \cup d''$, $d', d'' \in \Gamma_{\mathcal{A}}(n, n-v)$. On the other hand, if $Z \in \mathsf{iSets}(\mathfrak{v})$, it must be true that $\mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q') \neq \varnothing$ (if not, then $\deg \mathfrak{v} \geqslant n-2$ by property (A) of Theorem 8.2.5 for $\mathfrak{v}$ and this contradicts the size of Z). As $v-1 < n-2$, we can apply part 2 of the condition (B) of Theorem 8.2.5 to deduce that $Z = \overline{n} - b$ for a unique $b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q')$. Because $|Z| = v-1$, $|b| = n-v+1$. Now, $b \in \mathfrak{D}^{r+1}_{\mathcal{A}}(Q_l, Q') \subseteq \beta_{\mathcal{A}}(\mathfrak{s})$, thus there exists $X, X' \in \mathsf{iSets}(\mathfrak{s})$ such that $|X \cap b| = 1 \wedge |X' \cap b| = 1$ and this clearly implies that $|X| = |X'| = v$ and $\overline{n} - b = X \cap X'$, that is,

$$Z = \overline{n} - b = X \cap X',$$

and finally, we apply Λ_2 of the path $\mathfrak{s}$ to X, X' to find two unique boxes $c, c' \in \Gamma_{\mathcal{A}}(n, n-v)$ with $X = \overline{n} - c \wedge X' = \overline{n} - c'$, so that

$$\overline{n} - b = X \cap X' = (\overline{n} - c) \cap (\overline{n} - c') = \overline{n} - (c \cup c'),$$

then $b = c \cup c'$ and condition Ψ_3 is satisfied by $\mathfrak{q}$.

Ψ_4 The path $\mathfrak{q}$ fulfills this property because of the induction hypothesis on $\mathfrak{q}'$, condition Λ_2 for $\mathfrak{s}$ and (B) of Theorem 8.2.5 for $\mathfrak{v}$.

Ψ_5 Let $l \in \{v, \ldots, s\}$. This property is satisfied by $\mathfrak{q}$ if $\beta_{\mathcal{A}}(\mathfrak{q}; n-l+1) = \varnothing$. Suppose that $c \in \beta_{\mathcal{A}}(\mathfrak{q}; n-l+1)$, there exist $X, Y \in \mathsf{iSets}(\mathfrak{q})$ such that $|X \cap c| = |Y \cap c| = 1$. Then $|X| = |Y| = l$, so that we use property Ψ_3 (or Ψ_4) on X, Y to find two boxes $b_X, b_Y \in \beta_{\mathcal{A}}(\mathfrak{s}; n-l)$ such that $X = \overline{n} - b_X \wedge Y = \overline{n} - b_Y$. Also, $X \cap Y = \overline{n} - c$, thus $\overline{n} - c = X \cap Y = (\overline{n} - b_X) \cap (\overline{n} - b_Y) = \overline{n} - (b_X \cup b_Y)$. Therefore $c = b_X \cup b_Y$ and this says that $c \in \zeta(\beta_{\mathcal{A}}(\mathfrak{s}; n-l))$. Condition Ψ_5 is fulfilled by $\mathfrak{q}$.

We have shown with induction that we can build the path $\mathfrak{q}$ from the path $\mathfrak{s}$, no matter how many states $\mathfrak{s}$ has. This proves the Lemma. □

The following Lemma is a weaker version of Lemma 9.3.1, and it can be proven mostly as a consequence of that result.

Corollary 9.3.2. *Let $n \geqslant 3$ and $1 \leqslant s \leqslant n-2$ be fixed. Suppose that $\mathcal{A}$ is an iterated protocol with safe-consensus objects and there exists a path*

$$\mathfrak{s}\colon S_0 \overset{X_1}{\sim} \cdots \overset{X_r}{\sim} S_q \quad (q \geqslant 1),$$

of connected states of round $r \geqslant 0$ such that $\deg \mathfrak{s} \geqslant s$ and $\overline{n} \notin \beta_{\mathcal{A}}(\mathfrak{s})$. Then in round $r+1$ there exists a path

$$\mathfrak{q}\colon Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_u}{\sim} Q_u,$$

such that the following statements hold:

Ψ_1 *If* $\beta_{\mathcal{A}}(\mathfrak{s}; n-s+1) = \varnothing$, *then* $\deg \mathfrak{q} \geqslant s$.

Ψ_2 *If* $\beta_{\mathcal{A}}(\mathfrak{s}; n-s+1)$ *is not empty, then* $\deg \mathfrak{q} \geqslant s-1$.

Ψ_3 *For every* $Z \in \mathsf{iSets}(\mathfrak{q})$ *with* $|Z| = s-1$, *there exist* $X, X' \in \mathsf{iSets}(\mathfrak{s})$ *and a unique* $b \in \beta_{\mathcal{A}}(\mathfrak{s}; n-s+1)$ *such that* $Z = \overline{n} - b = X \cap X'$.

Ψ_5 $\beta_{\mathcal{A}}(\mathfrak{q}; n-s+2) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{s}; n-s+1))$.

We have gathered all the required elements to prove Theorem 9.3.3. This result uses the properties of Lemma 9.3.1 to find an infinite sequence of paths of connected reachable states, all these paths enjoy many of the properties of the resulting path of Lemma 9.3.1. To prove Theorem 9.3.3, we need to apply Theorem 9.1.10 from Section 9.1.

Theorem 9.3.3. *Let* $n \geqslant 3$ *and* $3 \leqslant m \leqslant n$ *be fixed. Suppose that* $\mathcal{A}$ *is an iterated protocol with safe-consensus objects such that* $\nu_{\mathcal{A}}(n,m) \leqslant n-m$. *If* S^r, Q^r *are two reachable states in* $\mathcal{A}$ *for some round* $r \geqslant 0$, *connected with a path* $\mathfrak{q}\colon S^r \sim \cdots \sim Q^r$ *such that* $\deg \mathfrak{q} \geqslant n-m+1$, *then for all* $u \geqslant 0$, *there exist successor states* S^{r+u}, Q^{r+u} *of* S^r *and* Q^r *respectively, in round* $r+u$ *of* $\mathcal{A}$, *such that* S^{r+u} *and* Q^{r+u} *are connected.*

Proof. Let $\mathcal{A}$ be a protocol with the given hypothesis, set $\mathfrak{q}_0 = \mathfrak{q}$ and $m = n-s+1$. Because $3 \leqslant m \leqslant n$, $s \in \{1, \ldots, n-2\}$. Using the path $\mathfrak{q}_0$ and applying Corollary 9.3.2, we build a path $\mathfrak{q}_1\colon S^{r+1} \sim \cdots \sim Q^{r+1}$, connecting the successor states S^{r+1}, Q^{r+1} of S^r and Q^r respectively, such that

$\Psi_{1,1}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_0; n-s+1) = \varnothing$, then $\deg \mathfrak{q}_1 > s-1$.

$\Psi_{1,2}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_0; n-s+1)$ is not empty, then $\deg \mathfrak{q}_1 \geqslant s-1$.

$\Psi_{1,3}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_1)$ with $|Z| = s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_0)$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_0; n-s+1)$ such that $Z = \overline{n} - b = X \cap X'$.

$\Psi_{1,5}$ $\beta_{\mathcal{A}}(\mathfrak{q}_1; n-s+2) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{q}_0; n-s+1))$.

Starting from $\mathfrak{q}_1$ and using induction together with Lemma 9.3.1, we can prove that for each $u \in \{1, \ldots, s-1\}$, there exist successor states S^{r+u}, Q^{r+u} of the states S^r and Q^r respectively, and a path

$$\mathfrak{q}_u\colon S^{r+u} \sim \cdots \sim Q^{r+u},$$

satisfying the properties:

$\Psi_{u,1}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-s+u) = \varnothing$, then $\deg \mathfrak{q}_u > s-u$.

$\Psi_{u,2}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-s+u)$ is not empty, then $\deg \mathfrak{q}_u \geqslant s-u$.

$\Psi_{u,3}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_u)$ with $|Z| = s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{u-1})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-s+1)$ such that $Z = \overline{n} - b = X \cap X'$.

$\Psi_{u,4}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_u)$ with $s-u \leqslant |Z| < s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{u-1})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-|Z|)$ such that $Z = \overline{n} - b = X \cap X'$ and $b = c_1 \cup c_2$, $c_k \in \beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-|Z|-1)$.

$\Psi_{u,5}$ $\beta_{\mathcal{A}}(\mathfrak{q}_u; n-l+1) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{q}_{u-1}; n-l))$ for $s-u \leqslant l < s$.

When $u = s-1$, we obtain a path $\mathfrak{q}_{s-1}\colon S^{r+s-1} \sim \cdots \sim Q^{r+s-1}$ that connects the states S^{r+s-1} and Q^{r+s-1}, such that

$\Psi_{s-1,1}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-1) = \varnothing$, then $\deg \mathfrak{q}_{s-1} > 1$.

$\Psi_{s-1,2}$ If $\beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-1)$ is not empty, then $\deg \mathfrak{q}_{s-1} \geqslant 1$.

$\Psi_{s-1,3}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_{s-1})$ with $|Z| = s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{s-2})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-s+1)$ such that $Z = \overline{n} - b = X \cap X'$.

$\Psi_{s-1,4}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_{s-1})$ with $1 \leqslant |Z| < s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{s-2})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-|Z|)$ such that $Z = \overline{n} - b = X \cap X'$ and $b = c_1 \cup c_2$, $c_k \in \beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-|Z|-1)$.

$\Psi_{s-1,5}$ $\beta_{\mathcal{A}}(\mathfrak{q}_{s-1}; n-l+1) \subseteq \zeta(\beta_{\mathcal{A}}(\mathfrak{q}_{s-2}; n-l))$ for $1 \leqslant l < s$.

Our final goal is to show that for all $v \geqslant s-1$, we can connect in each round $r+v$ of $\mathcal{A}$, successor states of S^r and Q^r. We first claim that for any $w = 0, \dots, s-1$ and $z \geqslant 0$,

$$\zeta^z(\beta_{\mathcal{A}}(\mathfrak{q}_w; n-s+1)) \subseteq \zeta^z(\Gamma_{\mathcal{A}}(n, n-s+1)),$$

(this is true because ζ preserves $\subseteq$) and combining this fact with the properties $\Psi_{u,5}$ and induction, we can show that for $1 \leqslant l < s$

$$\beta_{\mathcal{A}}(\mathfrak{q}_{s-1}; n-l+1) \subseteq \zeta^{s-l}(\beta_{\mathcal{A}}(\mathfrak{q}_{l-1}; n-s+1)) \subseteq \zeta^{s-l}(\Gamma_{\mathcal{A}}(n, n-s+1)). \tag{9.13}$$

And because $|\Gamma_{\mathcal{A}}(n, n-s+1)| = \nu_{\mathcal{A}}(n, n-s+1) \leqslant s-1$ and $m = n-s+1$, we use Theorem 9.1.10 to check that $\zeta^{s-1}(\Gamma_{\mathcal{A}}(n, n-s+1)) = \varnothing$, implying that

$$\beta_{\mathcal{A}}(\mathfrak{q}_{s-1}; n) = \varnothing.$$

With all this data, Lemma 9.3.1 and an easy inductive argument (starting at the base case $v = s-1$), we can find for all $v \geqslant s-1$, states S^{r+v}, Q^{r+v} of round $r+v$ of $\mathcal{A}$, which are successor states of S^r and Q^r respectively and connected with a path $\mathfrak{q}_v$ such that

$\Psi'_{v,1}$ $\deg \mathfrak{q}_v \geqslant 1$.

$\Psi'_{v,2}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_v)$ with $|Z| = s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{v-1})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{v-1}; n-s+1)$ such that $Z = \overline{n} - b = X \cap X'$.

$\Psi'_{v,3}$ For every $Z \in \mathsf{iSets}(\mathfrak{q}_v)$ with $1 \leqslant |Z| < s-1$, there exist $X, X' \in \mathsf{iSets}(\mathfrak{q}_{v-1})$ and a unique $b \in \beta_{\mathcal{A}}(\mathfrak{q}_{v-1}; n-|Z|)$ such that $Z = \overline{n} - b = X \cap X'$ and $b = c_1 \cup c_2$, $c_k \in \beta_{\mathcal{A}}(\mathfrak{q}_{v-1}; n-|Z|-1)$.

$\Psi'_{v,4}$ $\beta_{\mathcal{A}}(\mathfrak{q}_v; n-l+1) \subseteq \zeta^{s-l}(\Gamma_{\mathcal{A}}(n, n-s+1))$ for $1 \leqslant l < s$.

It is precisely these four properties of the path $\mathfrak{q}_v$ which allow us to find the path $\mathfrak{q}_{v+1}$, applying Lemma 9.3.1, such that $\mathfrak{q}_{v+1}$ enjoys the same properties. Therefore, starting from round r, we can connect in all rounds of $\mathcal{A}$, successor states of S^r and Q^r. The Theorem is proven. □

9.3.2 Results for $m = 2$

Now we will prove some technical results which cover the last case $m = 2$. The main result here is Theorem 9.3.5. Our strategy is basically the same that we used for the previous case. Given an iterated protocol with safe-consensus and a path of connected states with some properties, we prove the existence of a new path in the next iteration of $\mathcal{A}$, connecting successor states of the states from the first path. Then we show that if $\nu_{\mathcal{A}}(n, 2) \leqslant n - 2$, we can build an infinite sequence of paths of connected reachable states. It is in this section where Lemma 9.1.6 comes into play. Our first lemma is very simlar to Lemma 9.3.1.

Lemma 9.3.4. *Let $\mathcal{A}$ be an iterated protocol with safe-consensus objects. Suppose that there exists a partition $\overline{n} = A \cup B$ and a path*

$$\mathfrak{p}\colon S_0 \sim \cdots \sim S_l \qquad (l \geqslant 0)$$

of connected states in round $r \geqslant 0$ of $\mathcal{A}$, with the following properties

I) $\mathsf{iSets}(\mathfrak{p}) = \{A, B\}$;

II) $(\forall b \in \Gamma_{\mathcal{A}}(n, 2))(b \subseteq A \vee b \subseteq B)$.

Then in round $r + 1$ of $\mathcal{A}$ there exists a path

$$\mathfrak{q}\colon Q_0 \sim \cdots \sim Q_s \qquad (s \geqslant 1)$$

of connected states and the following properties hold:

a) Each state Q_k is of the form $Q_k = S_j \cdot \xi(X)$, where $X = A \vee X = B$;

b) $\mathsf{iSets}(\mathfrak{q}) = \{A, B\}$.

Proof. This proof is very similar in spirit to the proof of Lemma 9.3.1, so we omit it. □

Theorem 9.3.5. *Let $n \geqslant 2$. If $\mathcal{A}$ is an iterated protocol for n processes using safe-consensus objects with $\nu_{\mathcal{A}}(n, 2) \leqslant n - 2$ and S is a reachable state in $\mathcal{A}$ for some round $r \geqslant 0$, then there exists a partition of the set $\overline{n} = A \cup B$ such that for all $u \geqslant 0$, the states $S \cdot \xi^u(A)$ and $S \cdot \xi^u(B)$ are connected.*

Proof. This proof is analogous to the proof of Theorem 9.3.3. We use Lemma 9.1.6 to find the partition of $\overline{n}$ and then we apply induction combined with Lemma 9.3.4. We omit the details. □

9.3.3 The lower bound proof

With Theorems 9.3.3 and 9.3.5 at hand, the stage is set to prove the principal result of the second part of this thesis: The tight lower bound on the number of safe-consensus objects needed to solve the consensus task, with any iterated protocol with safe-consensus shared objects. The lower bound will be an easy corollary of Theorem 9.3.6, which says that any iterated protocol with safe-consensus for n-consensus, must provide the processes with at least $n - m + 1$ m-boxes for each $m \in \{2, \ldots, n\}$, i.e., the processes must be able to invoke safe-consensus shared objects in groups of size m in at least $n - m + 1$ different manners.

Theorem 9.3.6. *Let $n \geqslant 2$. If $\mathcal{A}$ is an iterated protocol for n-consensus using safe-consensus objects, then for every $m \in \{2, \ldots, n\}$, $\nu_{\mathcal{A}}(n, m) > n - m$.*

Proof. Assume that there exists a protocol $\mathcal{A}$ for consensus such that there is some m with $2 \leqslant m \leqslant n$ with $\nu_{\mathcal{A}}(n, m) \leqslant n - m$. Let O, U be the initial states in which all processes have as input values 0s and 1s respectively. We now find successor states of O and U in each round $r \geqslant 0$, which are connected. We have cases:

Case $m = 2$. By Theorem 9.3.5, there exists a partition of $\overline{n} = A \cup B$ such that for any state S and any $r \geqslant 0$, $S \cdot \xi^r(A)$ and $S \cdot \xi^r(B)$ are connected. Let OU be the initial state in which all processes with ids in A have as input value 0s and all processes with ids in B have as input values 1s. Then for all $r \geqslant 0$ we have that

$$O \cdot \xi^r(A) \overset{A}{\sim} OU \cdot \xi^r(A) \quad \text{and} \quad OU \cdot \xi^r(B) \overset{B}{\sim} U \cdot \xi^r(B)$$

and the states $OU \cdot \xi^r(A)$ and $OU \cdot \xi^r(B)$ are connected. Thus, for any r-round partial execution of $\mathcal{A}$, we can connect the states $O^r = O \cdot \xi^r(A)$ and $U^r = U \cdot \xi^r(B)$.

Case $3 \leqslant m \leqslant n$. It is not hard to see that using Lemma 5.4.3, we can connect O and U with a path $\mathfrak{q}$ of initial states of $\mathcal{A}$, such that $\deg \mathfrak{q} \geqslant n - 1 \geqslant n - m + 1$. By Theorem 9.3.3, for each round $r \geqslant 0$ of $\mathcal{A}$, there exist successor states O^r, U^r of O and U respectively, such that O^r and U^r are connected.

In this way, we have connected successor states of O and U in each round of the protocol $\mathcal{A}$. Because $\mathcal{A}$ is a protocol that solves consensus, there exists u such that U^u, O^u are connected decision states in round u of $\mathcal{A}$. Since O is a 0-valent state and O^u is a successor state of O, all processes decide 0 in O^u; and for similar reasons, all processes decide 1 in U^u. But the states U^u and O^u are connected, thus if processes decide 0 in O^u, they must decide 0 in U^u, so we have reached a contradiction. Therefore $\nu_{\mathcal{A}}(n, m) > n - m$. □

Theorem 9.3.6 gives us precise information about the number of different invocations of safe-consensus shared objects that the processes must perform when they are executing an iterated protocol to solve the consensus task. This result is far stronger that proving that any iterated protocol $\mathcal{A}$ with safe-consensus must satisfy the inequality $\nu_{\mathcal{A}}(n) \geqslant \binom{n}{2}$ to be able to solve n-consensus. Theorem 9.3.6 tell us that all inequalities $\nu_{\mathcal{A}}(n, m) > n - m$ must be satisfied if the protocol $\mathcal{A}$ is to solve consensus. For example, if $n = 10$ and $\mathcal{A}$ is a protocol to solve the consensus task for ten processes, using safe-consensus shared objects, and such that

$\nu_{\mathcal{A}}(10, m) = \binom{10}{m}$ for $m \neq 7$;

$\nu_{\mathcal{A}}(10, 7) \leqslant 10 - 7 = 3$.

This protocol $\mathcal{A}$ has in total $\nu_{\mathcal{A}}(10) = \sum_{m=2}^{10} \nu_{\mathcal{A}}(10, m) = 45 + 120 + 210 + 252 + 210 + 3 + 45 + 10 + 1 = 896$ different ways to invoke safe-consensus shared objects !! The consensus protocol of Chapter 7 has only 45 boxes and is able to solve consensus. However, Theorem 9.3.6 allow us to conclude that $\mathcal{A}$ cannot solve the consensus task, because $\nu_{\mathcal{A}}(10, 7) \leqslant 10 - 7 = 3$. So that any iterated protocol that solves consensus, must satisfy all the inequalities given in Theorem 9.3.6. And the consensus protocol of Chapter 7 satisfies this result in the best way possible, as it is confirmed in the following corollary.

Corollary 9.3.7. *For $n \geqslant 2$ and any iterated protocol $\mathcal{A}$ that solves n-consensus with safe-consensus objects , $\nu_{\mathcal{A}}(n) \geqslant \binom{n}{2}$.*

Proof. Applying Theorem 9.3.6 we obtain the following

$$\nu_{\mathcal{A}}(n) = \sum_{m=2}^{n} \nu_{\mathcal{A}}(n,m) \geqslant \sum_{m=2}^{n} n-(m-1) = \frac{n(n-1)}{2} = \binom{n}{2}.$$

□

Therefore, the protocol of Chapter 7 which implements the consensus task is sharp and there cannot be an iterated protocol with safe-consensus objects that solves the consensus task for n processes with less that $\binom{n}{2}$ safe-consensus objects.

The conclusions for the second part of this thesis can be found in Chapter 10.

10 Conclusions

In this chapter, we gather all our conclusions about our research work, presenting a short review of each part of this thesis and some open problems and further work that can be done to extend our results.

In Section 10.1, we present our conclusions for the first part of this thesis and Section 10.2 contains our conclusions for the second part.

10.1 Computable manifolds

In the first part of this thesis, we have proposed a starting point to build a computable theory for topological manifolds, viewing computability as a structure that we impose to topological manifolds. We have provided the most basic constructs needed to give computable versions of standard definitions and also proved many useful results. We also studied computable functions between computable manifolds and defined computable submanifolds. Finally, we proved an embedding theorem for compact computably Haussdorf computable manifolds, which is a computable version of the result which states that every compact manifold embeds in some euclidean space.

10.1.1 Further work and open problems

Starting from our computable theory of manifolds, there are many steps that can be taken to enrich this theory, here is a small list of some possibilities.

- Investigate which computability properties of computable euclidean spaces generalizes to computable manifolds.
- Try to improve the dimension of the euclidean space $\mathbb{R}^q$ in Theorem 4.2.2.
- Study specific computable classes of manifolds (e.g. computability in C^r or piecewise lineal manifolds).
- Find out which computable separation axioms [Wei10] fulfill computable manifolds.
- Inquire the computability properties of spaces of continuous functions between computable manifolds.
- Prove computable versions of important results about manifolds (e.g. the product structure theorem [KS77, Rud01]).

10.1.2 Structures on topological manifolds and computability

The computable theory of topological manifolds that we present in this thesis could also be used as a new part of the "standard" theory of manifolds. It is known that the most popular structures (topological, smooth and piecewise linear) coincide in dimension at most three and they are different in dimensions at least four[1] [Ker60, KS77, Rud01]. Also, it was recently proven in [Man13] that in dimensions at least 5, topological manifolds cannot be triangulated as simplicial complexes. We have introduced computability as a structure that we impose to topological manifolds. Which is the relationship of computable structures with the classical structures and simplicial triangulations? An important open question (among many others) in this direction, concerning the computable theory and the standard one is the following: Does there exists a second countable topological manifold E such that E does not admit a computable structure ?

10.2 The iterated model extended with safe-consensus

In the second part of this thesis, we have described the iterated model of distributed computing, a useful tool to study and understand the behavior of distributed systems. Also, we described how the runs of protocol in this model can be represented with simplicial complexes and we presented some standard tools (i.e. connectivity of states, valency, etc.) commonly used to investigate distributed systems. And we described the set agreement task, one of the most important distributed problems.

10.2.1 The IFSC model, consensus and set agreement

With the basic iterated model at hand, we defined in detail an extension of this model, using the safe-consensus task of [AGL09]. To show to the reader what sort of results can be achieved with the extended iterated model with safe-consensus and the topological approach, we introduced a restricted version of the iterated model with safe-consensus, called the iterated shared memory with full safe-consensus objects (IFSC) model. We first showed how to analyze protocol complexes in the IFSC model, and we discovered that the protocol complexes of protocols in the IFSC model are (globally) disconnected. We explained why local connectivity between some pairs of vertices of the protocol complex is sufficient to prove that consensus is unsolvable in the IFSC model. Thus we see that connectivity of the protocol complex (which is related to the notion of non-distinguishable states) is indeed fundamental in the study of distributed computing.

But we also proved that in the new IFSC model, we can solve $(n, n-1)$-set agreement. As this problem cannot be solved in the basic iterated model [BG93a, SZ93, HS93], we can conclude that the IFSC model is indeed more powerful that the basic iterated model.

Even when the IFSC model is a simple extension of the basic iterated model, there are several questions which remain open. For example,

- Is the IFSC model equivalent (for task solvability) to the standard model of distributed computed extended with safe-consensus shared objects, with the restriction that each safe-consensus object is invoked by all processes ?
- Is the $(n, n-2)$-set agreement task solvable in the IFSC model ?

[1]For the case of dimension four, smooth and piecewise linear structures coincide, but there exists a topological 4-manifold which has no smooth or picewise linear structure.

We conjecture that the answer to the second question is no.

10.2.2 The iterated model with safe-consensus, the consensus task and the lower bound

After working with the IFSC model, we investigated the computational power of safe-consensus to solve the consensus task. In the extended iterated model with safe-consensus tasks, we constructed a protocol which can solve n-consensus using $\binom{n}{2}$ safe-consensus objects. To make our protocol more readable and easier to understand, we introduced a new group consensus problem: The g-2coalitions-consensus.

The most important result that we have obtained was Theorem 9.3.6, which describes the $\binom{n}{2}$ lower bound on the number of safe-consensus objects necessary to implement n-consensus in the iterated model with safe-consensus. The proof of this lower bound is somewhat complicated. At the very end, it is bivalency [FLP85], but in order to be able to use bivalency, an intricate connectivity argument must be developed, in which we relate the way that the processes invoke the safe-consensus shared objects with subgraphs of the Johnson graphs. As connectivity of graphs appear again in the scene for consensus, this suggest that topology will play an important role to understand better the behavior of protocols equipped with shared objects more powerful that read/write memory registers (similar conclusions can be obtained from the work of [GRH06]). The proofs developed to obtain the lower bound say that the connectivity of the topological structures given by the shared objects used by the processes, can affect the connectivity of the protocol complex. In the case of the lower bound proof (Theorem 9.3.6), which can be consider as a 1-dimensional topological case, the connectivity of the graphs build with the boxes representing invocations of safe-consensus objects, affected the connectivity of the paths of global states, it was the exploiting of this fact that allowed us to obtain our lower bound result.

Concerning the topology of protocols in the iterated model with safe-consensus, here are some open questions and further work that can be done to extend the results presented in this thesis.

- Is the iterated model extended with safe-consensus objects equivalent (for task solvability) to the wait-free standard model extended with safe-consensus objects ? This is a generalization of the IFSC case and is a highly non-trivial question. We believe that a topological approach like the one in [HR12] could be used to answer this question.
- Find a lower bound for $\nu_{\mathcal{A}}(n)$, for any protocol $\mathcal{A}$ in the iterated model with safe-consensus objects that can solve (n,k)-set agreement (e.g., generalize Theorem 9.3.6 to the case of the set agreement task).
- Find out the precise connectivity (in the topological sense) of protocol complexes of protocols in the iterated model with safe-consensus.

With this we conclude this work.

Bibliography

[AAD^{+}93] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, 1993.

[ABND^{+}90] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rudiger Reischuk. Renaming in an Asynchronous Environment. *Journal of the ACM*, July 1990.

[ADLS94] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41:122–152, January 1994.

[AGL^{+}08] Yehuda Afek, Iftah Gamzu, Irit Levy, Michael Merritt, and Gadi Taubenfeld. Group renaming. In *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems*, volume 5401 of *LNCS*, pages 58–72, Berlin, Heidelberg, 2008. Springer-Verlag.

[AGL09] Yehuda Afek, Eli Gafni, and Opher Lieber. Tight group renaming on groups of size g is equivalent to g-consensus. In *Proceedings of the 23rd international conference on Distributed computing (DISC'09)*, volume 5805 of *LNCS*, pages 111–126, Berlin, Heidelberg, 2009. Springer-Verlag.

[Ale98] P. S. Alexandrov. *Combinatorial topology. Vol. 1, 2 and 3*. Dover Publications Inc., Mineola, NY, 1998. Translated from the Russian, Reprint of the 1956, 1957 and 1960 translations.

[AR02] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM J. Comput.*, 31(4):1286–1313, 2002.

[Arm83] Mark A. Armstrong. *Basic Topology*. Springer-Verlag, 1983.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[BC70] F. Brickell and R. S. Clark. *Differentiable manifolds: An introduction*. Van Nostrand Reinhold Company London, 1970.

[BCS97] P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic complexity theory*. Grundlehren der mathematischen Wissenschaften. Springer, 1997.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer-Verlag, New York, 1998. With a foreword by Richard M. Karp.

[BG93a] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, New York, NY, USA, 1993. ACM.

[BG93b] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*, pages 41–51, New York, NY, USA, 1993. ACM.

[BG97] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 189–198, New York, NY, USA, 1997. ACM.

[BGLR01] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum. The BG distributed simulation algorithm. *Distrib. Comput.*, 14(3):127–146, 2001.

[BHW08] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms*, pages 425–491. Springer New York, 2008.

[BI14] Konrad Burnik and Zvonko Iljazovic. Computability of 1-manifolds. *Logical Methods in Computer Science*, 10(2), 2014.

[BM75] A. Borodin and I. Munro. *The computational complexity of algebraic and numeric problems*. Theory of computation series. American Elsevier Pub. Co., 1975.

[BMZ90] Ofer Biran, Shlomo Moran, and Shumuel Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *J. Algorithms*, 11:420–440, September 1990.

[Bol98] B. Bollobás. *Modern Graph Theory: Béla Bollobás*. Graduate Texts in Mathematics Series. Springer, 1998.

[BP03] Vasco Brattka and Gero Presser. Computability on subsets of metric spaces. *Theor. Comput. Sci.*, 305(1-3):43–76, 2003.

[Bra98] V. Brattka. *Recursive and Computable Operations over Topological Structures*. PhD Thesis. Fachbereich Informatik, FernUniversität Hagen, 1998.

[Bra03] Vasco Brattka. Computability over Topological Structures. In S. Barry Cooper and Sergey S. Goncharov, editors, *Computability and Models*, pages 93–136. Kluwer Academic Publishers, New York, 2003.

[Bra05] Mark Braverman. On the complexity of real functions. In *FOCS*, pages 155–164, 2005.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc. (N.S.)*, 21(1):1–46, 1989.

[BW99] Vasco Brattka and Klaus Weihrauch. Computability on subsets of euclidean space i: Closed and compact subsets. *Theor. Comput. Sci.*, 219(1-2):65–93, 1999.

[Cha93] Soma Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105:132–158, July 1993.

[CM09] Wesley Calvert and Russell Miller. Real computable manifolds and homotopy groups. In *Proceedings of the 8th International Conference on Unconventional Computation*, UC '09, pages 98–109, Berlin, Heidelberg, 2009. Springer-Verlag.

[CnHR12] Armando Castañeda, Maurice Herlihy, and Sergio Rajsbaum. An equivariance theorem with applications to renaming. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 133–144, Berlin, Heidelberg, 2012. Springer-Verlag.

[CnIRR12] Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Renaming is weaker than set agreement but for perfect renaming: A map of sub-consensus tasks. In *Proceedings of the 10th Latin American International Conference on Theoretical Informatics*, LATIN'12, pages 145–156, Berlin, Heidelberg, 2012. Springer-Verlag.

[CnR08] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 295–304, New York, NY, USA, 2008. ACM.

[CnR12a] Armando Castañeda and Sergio Rajsbaum. An inductive-style procedure for counting monochromatic simplexes of symmetric subdivisions with applications to distributed computing. *Electron. Notes Theor. Comput. Sci.*, 283:13–27, June 2012.

[CnR12b] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: The upper bound. *J. ACM*, 59(1):3:1–3:49, March 2012.

[Coo04] S.B. Cooper. *Computability theory*. Chapman & Hall/CRC mathematics. Chapman & Hall/CRC, 2004.

[CR12] Rodolfo Conde and Sergio Rajsbaum. An introduction to the topological theory of distributed computing with safe-consensus. *Electronic Notes in Theoretical Computer Science*, 283(0):29 – 51, 2012. Proceedings of the workshop on Geometric and Topological Methods in Computer Science (GETCO).

[CR14] Rodolfo Conde and Sergio Rajsbaum. The complexity gap between consensus and safe-consensus. In Magnús M. Halldórsson, editor, *Structural Information and Communication Complexity*, volume 8576 of *Lecture Notes in Computer Science*, pages 68–82. Springer International Publishing, 2014.

[CRR11] Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Computer Science Review*, 5(3):229–251, 2011.

[DL98] Philippe Delsarte and Vladimir I. Levenshtein. Association schemes and coding theory. *IEEE Transactions on Information Theory*, 44(6):2477–2504, 1998.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: Crash failures. *Information and Computation*, 88(2):156 – 186, 1990.

[Eng89] R. Engelking. *General topology*. volume 6 of Sigma series in pure mathematics. Heldermann, Berlin, 1989.

[Eng95] R. Engelking. *Theory of dimensions, finite and infinite*. Sigma series in pure mathematics. Heldermann, 1995.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[Fre82] Michael Hartley Freedman. The topology of four-dimensional manifolds. *J. Differential Geom.*, 17(3):357–453, 1982.

[Gaf09] Eli Gafni. The extended BG-simulation and the characterization of t-resiliency. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 85–92, New York, NY, USA, 2009. ACM.

[Gau74] D. B. Gauld. Topological properties of manifolds. *The American Mathematical Monthly*, 81(6):pp. 633–636, 1974.

[GR05] Eli Gafni and Sergio Rajsbaum. Musical benches. In *Proceedings of the 19th international conference on Distributed computing (DISC'05)*, volume 3724 of *LNCS*, pages 63–77, Berlin, Heidelberg, 2005. Springer-Verlag.

[GR10a] Eli Gafni and Sergio Rajsbaum. Distributed programming with tasks. In Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah, editors, *Principles of Distributed Systems*, volume 6490 of *Lecture Notes in Computer Science*, pages 205–218. Springer Berlin Heidelberg, 2010.

[GR10b] Eli Gafni and Sergio Rajsbaum. Recursion in distributed computing. In Shlomi Dolev, Jorge Cobb, Michael Fischer, and Moti Yung, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 6366 of *Lecture Notes in Computer Science*, pages 362–376. Springer Berlin Heidelberg, 2010.

[GRH06] Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus tasks: Renaming is weaker than set agreement. In *Proceedings of the 20th international conference on Distributed computing (DISC'06)*, volume 4167 of *LNCS*, pages 329–338, Berlin, Heidelberg, 2006. Springer-Verlag.

[Grz57] A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–71, 1957.

[GS80] David E. Galewski and Ronald J. Stern. Classification of simplicial triangulations of topological manifolds. *The Annals of Mathematics*, 111(1):pp. 1–34, 1980.

[GSW07] Tanja Grubba, Matthias Schröder, and Klaus Weihrauch. Computable metrization. *Mathematical Logic Quarterly*, 53(4-5):381–395, 2007.

[GW05] Tanja Grubba and Klaus Weihrauch. A computable version of dini's theorem for topological spaces. In *Proceedings of the 20th international conference on Computer and Information Sciences*, ISCIS'05, pages 927–936, Berlin, Heidelberg, 2005. Springer-Verlag.

[GWX08] Tanja Grubba, Klaus Weihrauch, and Yatao Xu. Effectivity on continuous functions in topological spaces. *Electronic Notes in Theoretical Computer Science*, 202(0):237 – 254, 2008. Proceedings of the Fourth International Conference on Computability and Complexity in Analysis (CCA 2007).

[Her91] Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991.

[Her10] Maurice Herlihy. Applications of shellable complexes to distributed computing. In *Proceedings of the 21st International Conference on Concurrency Theory*, CONCUR'10, pages 19–20, Berlin, Heidelberg, 2010. Springer-Verlag.

[HKR13] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[Hör90] L. Hörmander. *An introduction to complex analysis in several variables*. North-Holland mathematical library. North-Holland, 1990.

[HR03] Maurice Herlihy and Sergio Rajsbaum. A classification of wait-free loop agreement tasks. *Theoretical Computer Science*, 291(1):55 – 77, 2003.

[HR10a] Maurice Herlihy and Sergio Rajsbaum. Concurrent computing and shellable complexes. In *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pages 109–123, Berlin, Heidelberg, 2010. Springer-Verlag.

[HR10b] Maurice Herlihy and Sergio Rajsbaum. The topology of shared-memory adversaries. In *PODC '10: Proceeding of the 29th ACM symposium on Principles of distributed computing*, pages 105–113, New York, NY, USA, 2010. ACM.

[HR12] Maurice Herlihy and Sergio Rajsbaum. Simulations and reductions for colorless tasks. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 253–260, New York, NY, USA, 2012. ACM.

[HR13] Maurice Herlihy and Sergio Rajsbaum. The topology of distributed adversaries. *Distributed Computing*, 26(3):173–192, 2013.

[HRR12] Maurice Herlihy, Sergio Rajsbaum, and Michel Raynal. Computability in distributed computing: A tutorial. *SIGACT News*, 43(3):88–110, August 2012.

[HS93] Maurice Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 111–120, New York, NY, USA, 1993. ACM.

[HS99] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.

[HU90] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.

[Hud69] J.F.P. Hudson. *Piecewise linear topology*. Mathematics lecture note series. W. A. Benjamin, 1969.

[HW48] W. Hurewicz and H. Wallman. *Dimension theory*. Princeton mathematical series. Princeton University Press, 1948.

[HW90] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.

[Ilj13] Zvonko Iljazovic. Compact manifolds with computable boundaries. *Logical Methods in Computer Science*, 9(4), 2013.

[Ker60] Michel A. Kervaire. A manifold which does not admit any differentiable structure. *Comment. Math. Helv.*, 34:257–270, 1960.

[Ko91] Ker-I Ko. *Complexity theory of real functions*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1991.

[Koz97] D. Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997.

[KS77] R.C. Kirby and L. Siebenmann. *Foundational essays on topological manifolds, smoothings, and triangulations*. Annals of mathematics studies. Princeton University Press, 1977.

[Kur66] K. Kuratowski. *Topology*. Number v. 1 in Topology. Academic Press, 1966.

[KW85] Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoret. Comput. Sci.*, 38(1):35–53, 1985.

[LAA87] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Parallel and Distributed Computing, Advances in Computing Research. F. P. Preparata ed., JAI Press, Greenwich, CT*, 4:163–183, 1987.

[Lac55] Daniel Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles. II, III. *C. R. Acad. Sci. Paris*, 241:13–14, 151–153, 1955.

[Man13] C. Manolescu. Pin(2)-equivariant Seiberg-Witten Floer homology and the Triangulation Conjecture. *ArXiv e-prints*, March 2013.

[MR02] Yoram Moses and Sergio Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002.

[Mun00] J.R. Munkres. *Topology*. Prentice Hall, Incorporated, 2000.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in analysis and physics*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1989.

[Raj10] Sergio Rajsbaum. Iterated shared memory models. In Alejandro López-Ortiz, editor, *LATIN 2010: Theoretical Informatics*, volume 6034 of *Lecture Notes in Computer Science*, pages 407–416. Springer Berlin / Heidelberg, 2010.

[RRT08a] Sergio Rajsbaum, Michel Raynal, and Corentin Travers. An impossibility about failure detectors in the iterated immediate snapshot model. *Inf. Process. Lett.*, 108(3):160–164, 2008.

[RRT08b] Sergio Rajsbaum, Michel Raynal, and Corentin Travers. The iterated restricted immediate snapshot model. In *COCOON '08: Proceedings of the 14th annual international conference on Computing and Combinatorics*, pages 487–497, Berlin, Heidelberg, 2008. Springer-Verlag.

[RS82] C.P. Rourke and B.J. Sanderson. *Introduction to piecewise-linear topology*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, 1982.

[Rud01] Y. B. Rudyak. Piecewise linear structures on topological manifolds. *ArXiv Mathematics e-prints*, May 2001.

[RW13] Robert Rettinger and Klaus Weihrauch. Products of effective topological spaces and a uniformly computable tychonoff theorem. *Logical Methods in Computer Science*, 9(4), 2013.

[Sch02] M. Schröder. Extended admissibility. *Theoretical Computer Science*, 284(2):519 – 538, 2002.

[Sch03] M. Schröder. *Admissible representations for continuous computations*. Informatik-Berichte. Fernuniv., Fachbereich Informatik, 2003.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[Sma61] Stephen Smale. Generalized Poincaré's conjecture in dimensions greater than four. *Ann. of Math. (2)*, 74:391–406, 1961.

[SZ93] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: the topology of public knowledge. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 101–110, New York, NY, USA, 1993. ACM.

[SZ00] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[Tar48] A. Tarski. *A decision method for elementary algebra and geometry*. Rand report. Rand Corporation, 1948.

[Tur36] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[TW11] N. Tavana and K. Weihrauch. Turing machines on represented sets, a model of computation for Analysis. *ArXiv e-prints*, May 2011.

[Wei87] Klaus Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.

[Wei93] Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113(2):191 – 210, 1993.

[Wei00] K. Weihrauch. *Computable analysis: an introduction*. Texts in theoretical computer science. Springer, 2000.

[Wei08] Klaus Weihrauch. The computable multi-functions on multi-represented sets are closed under programming. *Journal of Universal Computer Science*, 14(6):801–844, mar 2008.

[Wei10] Klaus Weihrauch. Computable separation in topology, from t0 to t2. *Journal of Universal Computer Science*, 16(18):2733–2753, sep 2010.

[WG09] K. Weihrauch and T. Grubba. Elementary computable topology. *Journal of Universal Computer Science*, 15(6):1381–1422, 2009.

[Whi36] Hassler Whitney. Differentiable manifolds. *Ann. of Math.*, 37(3):pp. 645–680, 1936.

[Zho96] Qing Zhou. Computable real-valued functions on recursive open and closed subsets of euclidean space. *Mathematical Logic Quarterly*, 42(1):379–409, 1996.