Towards a Topological Characterization of Asynchronous Complexity

by

Gunnar Waaler Hoest

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 3, 1997

Author . . . .
Department of Electrical Engineering and Computer Science
September 3, 1997

Certified by . . . .
Nancy A. Lynch
Professor
Thesis Supervisor

Certified by . . . .
Nir Shavit
Visiting Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Towards a Topological Characterization of Asynchronous Complexity

by

Gunnar Waaler Hoest

Submitted to the Department of Electrical Engineering and Computer Science
on September 3, 1997, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In this thesis, we introduce the use of topological models and methods, formerly used to analyze computability problems in asynchronous, distributed computer systems, as tools for the quantification and classification of *complexity* in such systems.

We extend Herlihy and Shavit's topological computability model [26] to obtain a topological framework for modeling decision tasks and protocols in the *non-uniform iterated immediate snapshot (NIIS) model*, a generalization of the *iterated immediate snapshot (IIS) model* of Borowsky and Gafni [10]. We also present an *Asynchronous Complexity Theorem*, which gives a complete characterization of the complexity of solving decision tasks in this model. Our theorem states that the time complexity of any protocol in the non-uniform iterated immediate snapshot model solving a decision task is equal to the level of non-uniform chromatic subdivision necessary to allow a simplicial map from the task's input complex to its output complex.

To show the power of our theorem, we use it to prove tight upper and lower bounds on the time to achieve N process *Approximate Agreement*. Our bound of $\left\lfloor \log_d \frac{input-range}{\epsilon} \right\rfloor$ where $d = 3$ for two processes and $d = 2$ for three or more shows that the intriguing gap between the known lower and upper bounds implied by the work of Aspnes and Herlihy [1] is not a technical coincidence.

The main contribution of this work is that, using our framework, we can model and reason about concurrent executions of asynchronous, distributed protocols using simple, geometric and topological arguments that require no explicit mention of concurrency at all.

Thesis Supervisor: Nancy A. Lynch
Title: Professor

Thesis Supervisor: Nir Shavit
Title: Visiting Professor

Acknowledgments

First of all I would like to thank Nancy Lynch, Nir Shavit and the members of the Theory of Distributed Systems Group for their advice and support. I would also like to thank Serafim Batzoglou for valuable discussions about topological analysis of the Approximate Agreement problem. Thanks to Asli Aker for much needed support and comfort while working on this thesis. Finally, to my parents, Sigurd and Liv Hoest, and to my brother, Einar Hoest: *Takk for all deres støtte, hjelp og tålmodighet - nå er denne oppgaven endelig ferdig!*

Contents

List of Figures

Chapter 1

Introduction

As we enter the 21st century, computers are progressively being turned into interacting coordination devices in asynchronous, distributed systems. Unfortunately, the standard, Turing notions of computability and complexity are not sufficient for evaluating the behavior of such systems. In the last few years, techniques of modeling and analysis based on classical algebraic topology [4, 10, 12, 16, 20, 18, 19, 21, 22, 26, 29], in conjunction with distributed simulation methods [8, 9, 10, 23] have brought about significant progress in our understanding of *asynchronous computability* problems. In this thesis, these techniques are extended and modified to provide a framework for analyzing *asynchronous complexity* problems.

1.1 Historical Background and Related Work

In this section, we give a brief account of previous work on computability problems in fault-prone, asynchronous, distributed systems, applications of algebraic topology to asynchronous computability problems, simulation techniques, and also on characterizing the *Approximate Agreement* task.

In 1985, a fundamental paper by Fischer, Lynch and Paterson [15] demonstrated that traditional Turing computability theory is not sufficient for analyzing computability problems in asynchronous, distributed systems. In particular, it showed that the well-known *Consensus* task, in which each participating process has a private input value drawn from some set S, and every non-faulty process must decide on an output value equal to the input of some process, cannot be solved in a message passing system if only one process may fail by halting. Later, it was also shown that the message passing and shared memory models are equivalent [2], so this result carries over to shared memory systems as well. This fundamental discovery led to the creation of a highly active research area, which is surveyed in a recent book by Lynch [25].

In 1988, Biran, Moran and Saks [6] introduced a graph-theoretic framework that provides a complete characterization of the types of tasks that can be solved in a message passing or shared memory system in the presence of a single failure. This framework proved hard to extend to more than one failure, however, and even the problems of completely characterizing specific tasks such as *Renaming* [5] and *Set Agreement* [11] remained unsolved.

In 1993, three research teams working independently - Borowsky and Gafni [8], Saks and Zaharoglou [29], and Herlihy and Shavit [21], derived lower bounds for solving the *Set Agreement* task. The paper of Borowsky and Gafni introduced a powerful new simulation technique for proving solvability and unsolvability results in asynchronous, distributed systems. The technique allows N-process protocols to be executed by fewer processes in a resilient way, and has recently been proven correct by Lynch and Rajsbaum [23].

The landmark paper of Herlihy and Shavit [21] introduced a new formalism based on tools from classical, algebraic topology for reasoning about computations in asynchronous, distributed systems in which up to all but one process may fail. Their framework consists of modeling tasks and protocols using algebraic structures called *simplicial complexes*, and then applying standard homology theory to reason about them. In 1994 and 1995, Herlihy and Shavit extended this framework by providing the Asynchronous Computability Theorem, which states a condition that is necessary and sufficient for a task to be solvable by a wait-free protocol in shared memory [22, 26], and showed applications of this theorem to tasks such as *Set Agreement* and *Renaming*. In her PhD thesis, Elisabeth Borowsky generalized this solvability condition to a model consisting of regular shared memory augmented with set-consensus objects, and under more general resiliency requirements [7].

In 1993, Chaudhuri, Herlihy and Lynch [12] also used topological and geometric arguments to prove tight bounds on solving the *Set Agreement* problem in the *synchronous* message passing model where an arbitrary number of processes may fail.

In 1994, Herlihy and Rajsbaum derived further impossibility results for *Set Agreement* by applying classical homology theory [20]. Moreover, in a unifying paper in 1995, Herlihy and Rajsbaum provided a common, general framework for describing a wide collection of impossibility results by using chain maps and chain complexes [18]. At the same time, Attiya and Rajsbaum reproved several impossibility results using a purely combinatorial framework [4].

Gafni and Koutsopias recently used a reduction from the classical contractibility problem of algebraic topology to show that it is undecidable whether a certain class of 3-process tasks are wait-free solvable in the shared memory model or not [16]. This work was then generalized by Herlihy and Rajsbaum to arbitrary numbers of processes and failures in a variety of computational models. [19]. Recently, Havlicek showed that, while undecidability holds in the general case, the problem of solvability is in fact decidable for a relatively large class of tasks [17].

The immediate snapshot (IS) object was introduced by Borowsky and Gafni in 1993 [9]. It is the basic building block of the iterated immediate snapshot (IIS) model, first implicitly used by Herlihy and Shavit [21, 22], and more recently formulated as a computation model by Borowsky and Gafni [10] as part of their new, simplified proof of the Asynchronous Computability Theorem of Herlihy and Shavit [21, 22, 26]. This work also shows that the IIS model is computationally equivalent to standard shared memory models by providing a wait-free implementation of IIS from shared memory, and vice versa. It is not clear, however, whether these implementations are optimal from a complexity-theoretic viewpoint.

The *Approximate Agreement* problem, a weakening of the *Consensus* problem in which each process has a real valued input, and the non-faulty processes must agree on output values at most $\epsilon > 0$ apart. It was first introduced in 1986 by Dolev, Lynch, Pinter, Stark and Weihl [13], in a paper showing that this task can be solved in both the synchronous and asynchronous message passing models even when assuming the Byzantine failure model (in which processes may exhibit arbitrary, even malicious behavior). The paper also provided matching upper and lower bounds for solving the task in these settings. These results were extended to various failure models by Fekete [14], who also showed optimality in terms of the number of rounds of communication used.

In 1994, Attiya, Lynch and Shavit published a paper giving a $\Omega(\log n)$ lower bound, together with an almost matching $O(\log n)$ upper bound, where n is the number of processes, on solving *Approximate Agreement* in failure-free single-writer, multi-reader shared memory systems [3]. These results were part of a proof that, in certain settings, wait-free algorithms are inherently *slower* than non-wait-free algorithms.

This work was extended by Schenk [30], who showed matching upper and lower bounds for solving the task in the asynchronous single-writer, multi-reader shared memory model where the magnitudes of the inputs are bounded above.

Finally, in 1994, Aspnes and Herlihy [1] showed a $\left\lfloor \log_3 \frac{input-range}{\epsilon} \right\rfloor$ lower bound, together with a $\left\lfloor \log_2 \frac{input-range}{\epsilon} \right\rfloor$ upper bound on solving *Approximate Agreement* using wait-free protocols in the atomic snapshot shared memory model.

1.2 This Work

In this thesis, we study the problem of analyzing the complexity of wait-free shared memory protocols solving *decision tasks*. In such a task, each process starts with a private *input* value and must decide on a private *output* value according to some task specification.

We focus on a generalized version of Borowsky and Gafni's iterated immediate snapshot (IIS) model [10], called the non-uniform iterated immediate snapshot (NIIS) Model. The IIS model has already been used successfully by Borowsky and Gafni [10] as part of their new simplified proof of the asynchronous computability theorem [26]. We believe it is a good first candidate for topological modeling and analysis, since it has a particularly nice and regular geometric representation.

Keeping in style with Herlihy and Shavit's topological computability framework and Asynchronous Computability Theorem [26], we show in Theorem 4.2 that there is a wait-free protocol in the NIIS model solving a given decision task with complexity k on a set of inputs, where k is an integer, if and only if there is a non-uniform chromatic subdivision of the input complex with level k on the corresponding input simplex that can be mapped simplicially to the output complex in accordance with the task specification.

The non-uniform chromatic subdivisions we introduce are a looser and more general form of the iterated standard chromatic subdivisions used in the computability work of Herlihy and Shavit and Borowsky and Gafni [21, 22, 26, 10]. Unlike iterated standard chromatic subdivisions, non-uniform chromatic subdivisions allow individual simplexes in a complex to be subdivided different numbers of times, while assuring that the subdivision of the complex as a whole remains consistent.

Non-uniformity is a useful property when analyzing complexity, since the time complexity, and hence the level of subdivision, of an input simplex may differ from one set of inputs to the next. Considering just the complexity of the worst case execution over all input sets would in many cases make a complexity theorem less than useful.

The power of Theorem 4.2 lies in its ability to allow one to reason about the complexity of solving decision tasks in a purely topological setting. As we will show, the non-uniform chromatic subdivisions of a complex provide a clean and high level way of thinking about the multitude of concurrent executions of a NIIS protocol. We found this topological representation very helpful, and we are sure that it will prove to be an invaluable tool for designing and analyzing concurrent protocols.

We provide one example application of Theorem 4.2. In Chapter 5, we use our topological framework to show tight upper and lower bounds on the time to solve the *Approximate Agreement* problem wait-free in the NIIS model. The best known previous results, due to Aspnes and Herlihy [1], imply an $\left\lfloor \log_2 \frac{input-range}{\epsilon} \right\rfloor$ upper bound and an $\left\lfloor \log_3 \frac{input-range}{\epsilon} \right\rfloor$ lower bound. We close this gap, proving matching upper and lower bounds of $\left\lfloor \log_d \frac{input-range}{\epsilon} \right\rfloor$ where $d = 3$ for two processes and $d = 2$ for three or more.

1.3 Organization

The thesis is organized as follows. Chapter 2 provides a formal definition of decision tasks. It also contains a thorough description of our model of computation, together with the complexity measures we use for analyzing protocols in this model. Chapter 3 contains a collection of necessary definitions and results from algebraic topology, as well as a description of how we model decision tasks and NIIS protocols topologically. It also contains definitions of the standard chromatic subdivision and the non-uniform chromatic subdivision. Chapter 4 contains a statement and proof our main theorem. Chapter 5 contains an application of our Asynchronous Complexity Theorem to the *Approximate Agreement* task. Finally, Chapter 6 summarizes our results, and also gives some directions for further research.

Chapter 2

Model

In order to develop a useful and applicable complexity theory for asynchronous, distributed computer systems, we need to define some reasonable model of such systems. This model must be detailed enough so as to accurately and faithfully capture the inherent complexity of solving tasks in real distributed systems, yet be simple enough so as to easily lend itself to some practical form of complexity analysis. The model we consider in this thesis consists of a class of one-shot distributed problems, called *decision tasks*, together with a novel model of computation, a type of shared memory called the *non-uniform iterated immediate snapshot (NIIS) model*. This chapter contains a detailed description of these fundamental concepts. It also contains the complexity measures that will be used to analyze the complexity of solving decision tasks in the non-uniform iterated immediate snapshot model.

2.1 Informal Synopsis

We begin with an informal synopsis of our model, which largely follows that of Herlihy and Shavit [21, 22, 26]. Some fixed number $N = n + 1$ of sequential threads of control, called *processes*, communicate by asynchronously accessing shared memory in order to solve *decision tasks*. In such a task, each process starts with a private *input* value and halts with a private *output* value. For example, in the well-known *Binary Consensus* task, the processes have binary inputs, and must agree on some process's input [15]. A *protocol* is a distributed program that solves a decision task in such a system. A protocol is *wait-free* if it guarantees that every non-faulty process will halt in a finite number of steps, independent of the progress of the other processes. The *time complexity* of solving a decision task in this model on a given input set is the supremum of the number of accesses to shared memory made by any process on that input set.

2.2 Decision Tasks

In this section, we define decision tasks more precisely. This class of tasks is intended to provide a simple model of reactive systems, such as databases, file systems, or automatic teller machines. An input value represents information entering the system from the surrounding environment, such as a character typed at a keyboard, a message from another computer, or a signal from a sensor. An output value models an effect on the outside world, such as an irrevocable decision to commit a transaction, to dispense cash, or to launch a missile. Informally speaking, A decision task is a relation between vectors of input values and vectors of output values. We define this more precisely below.

Let D_I and D_O be two data types, possibly identical, called the *input data type* and the *output data type*, respectively. We first define the concept of an input vector.

Definition 2.1 *An $n+1$-process* input vector $\vec{I}$ *is an $n+1$-dimensional vector, indexed by $\mathbb{Z}_{n+1}$ (the integers $\bmod n$), each component of which is either an object of type D_I, or the distinguished value $\perp$, with the additional requirement that at least one component of $\vec{I}$ must be different from $\perp$.*

The definition of output vectors is similar to that for input vectors:

Definition 2.2 *An $n+1$-process* output vector $\vec{O}$ *is an $n+1$-dimensional vector, indexed by $\mathbb{Z}_{n+1}$, each component of which is either an object of type D_O, or the distinguished value $\perp$.*

When it is clear from the context, we omit mentioning the number of processes in specifying input and output vectors. We denote the i-th component of an input vector $\vec{I}$ by $\vec{I}[i]$, and similarly, we denote the i-th component of an output vector $\vec{O}$ by $\vec{O}[i]$. In the remainder of this thesis, unless stated otherwise, we will assume that i and j are index values in the set $\mathbb{Z}_{n+1}$. These index values will be used both for specifying vector elements and also to index processes. We note that, in this thesis, we will use the terms "one-dimensional array"("array" for short) and "vector" interchangeably - they refer to the same data type.

Definition 2.3 *A vector $\vec{U}$ is a* prefix *of $\vec{V}$ if $\vec{V}[i] = \perp$ implies that $\vec{U}[i] = \perp$, and for all i such that $\vec{U}[i] \neq \perp$, $\vec{U}[i] = \vec{V}[i]$.*

Definition 2.4 *A set V of vectors is* prefix-closed *if for all $\vec{V} \in V$, every prefix $\vec{U}$ of $\vec{V}$ is in V.*

In this thesis, we will only consider sets of input and output vectors that are finite and prefix-closed.

Definition 2.5 *An* input set *is a finite, prefix-closed set of input vectors. An* output set *is a finite, prefix-closed set of output vectors.*

Next, we define the notion of a task specification map, which maps each element of the input set to a subset of the output set. Our definition is similar to that of Havlicek [17].

Definition 2.6 *Let I and O be input and output sets, respectively. A* task specification map *relating the two sets is a relation $\gamma \subseteq I \times O$ such that the following conditions hold:*

- *For all $\vec{I} \in I$, there exists a vector $\vec{O} \in O$ such that $(\vec{I}, \vec{O}) \in \gamma$.*
- *For all $(\vec{I}, \vec{O}) \in \gamma$, and for all i, $\vec{I}[i] = \perp$ if and only if $\vec{O}[i] = \perp$.*

As a convenient notation, we denote the set of vectors $\vec{O}$ in O such that $(\vec{I}, \vec{O}) \in \gamma$ by $\gamma(\vec{I})$. For a given input vector $\vec{I}$, the set of vectors $\gamma(\vec{I})$ simply represents the set of legitimate output vectors for the set of inputs specified by $\vec{I}$. This set will generally contain more than one allowable output vector. We are now ready to give a precise definition of decision tasks.

Definition 2.7 *A* decision task *$\mathcal{D} = \langle I, O, \gamma \rangle$ is a tuple consisting of a set I of input vectors, a set O of output vectors, and a task specification map γ relating these two sets.*

We note that, by definition, decision tasks are inherently *one-shot*, in the sense that all processes have a single input and must decide on a single output exactly once.

Not all entries in a given input vector need contain an input value; some may contain the special value $\perp$, indicating that some processes do not receive an input value. We formalize this notion of participation in the definition below.

Definition 2.8 *For any input vector $\vec{I}$, if the i-th component is not $\perp$, then i* participates *in $\vec{I}$. Otherwise, we say that i does not participate in $\vec{I}$. Moreover, we define the* participating set *in $\vec{I}$ to be set of participating indexes.*

As noted by Herlihy and Shavit [21, 22, 26], the reason for incorporating an explicit notion of participating indexes in our formalism for decision tasks is that it is convenient for distinguishing between tasks such as the following, which have the same sets of input and output vectors, but different task specification maps.

Example 2.9 *The $n+1$-process* Unique-Id *task is defined as follows:*

- $I = \{[x_0, \ldots, x_n] \mid x_i \in \{i, \perp\}\}$.
- $O = \{[x_0, \ldots, x_n] \mid x_i \in \mathbb{Z}_{n+1} \cup \{\perp\}, (x_i = x_j) \Rightarrow (x_i = \perp)\}$.
- $\gamma = \{(\vec{I}, \vec{O}) \mid (\vec{I}[i] = \perp \Leftrightarrow \vec{O}[i] = \perp)\}$.

Example 2.10 *The $n+1$-process* Fetch-And-Increment *task is defined as follows:*

- $I = \{[x_0, \ldots, x_n] \mid x_i \in \{i, \perp\}\}$.
- $O = \{[x_0, \ldots, x_n] \mid x_i \in \mathbb{Z}_{n+1} \cup \{\perp\}, (x_i = x_j) \Rightarrow (x_i = \perp)\}$.
- *γ is the set of pairs $(\vec{I}, \vec{O})$ for which the following conditions hold:*
  - $\vec{I}[i] = \perp \Leftrightarrow \vec{O}[i] = \perp$
  - *If $\vec{O}[i] \neq \perp$, then $\vec{O}[i]$ must be less than the size of the participating set of $\vec{I}$.*

The two decision tasks above differ in that the task specification map of *Fetch-And-Increment* involves explicit mention of the participating set of the input vectors, while that of *Unique-Id* does not.

2.3 One-Shot Immediate Snapshot

Borowsky and Gafni's *immediate snapshot (IS) object* [9] has by now been proven to be a useful building block for the construction and analysis of protocols in asynchronous, distributed systems [9, 10, 21, 22, 26, 28].

Informally, an $n+1$-process IS object consists of a shared $n+1$-dimensional memory array, and supports a single type of external operation, called *writeread*. The IS object has $n+1$ ports, one for each process. We typically associate process i with the i-th port. Each *writeread* operation writes a value to a single shared memory array cell, and then immediately returns a snapshot view of the entire array. There is a separate *writeread* operation for each of the IS object's $n+1$ ports. A *writeread* operation on port i writes its value to the i-th cell of the memory array.

Formally, we can specify IS objects as I/O automata [24]. Let D be any data type, and define $\vartheta(D)$ to be the data type $(D \cup \{\perp\})^{n+1}$, the set of all $n+1$-arrays each of whose cells contains either an element of D, or $\perp$. We index the elements of $\vartheta(D)$ using the numbers in $\mathbb{Z}_{n+1}$. An IS automaton for $n+1$-processes and data type D called IS_x is defined as in Figure 2-1. We refer to such an object as an IS_D^{n+1} object. When the data type and number of processes are clear from the context, we usually omit the sub- and superscripts above.

For all i, the *inv_writeread(v)*$_{i,x}$ action simply writes the input value v to the i-th cell of the *input_value* array of IS_x. This array provides temporary storage for inputs to the IS_D^{n+1} object. At the same time, the flag *"inv"* is written to the i-th cell of the *interface* array, which indicates that an input has arrived on port i. The *update($\mathcal{U}$)* action periodically copies a set of values corresponding to the indexes in $\mathcal{U}$, from the *input_value* array to the *memory* array. The set $\mathcal{U}$ must be a subset of the indexes i with the property that *interface[i]* $=$ *inv*. Additionally, a copy of the *memory* array is written to the i-th cell of the *return_value* array for each $i \in \mathcal{U}$. Finally, the flag *"ret"* is written to the i-th cell of the *interface* array for each $i \in \mathcal{U}$, indicating that a response value to the invocation on port i is available. The *ret_writeread(S)*$_{i,x}$ output action provides a response to a previous invocation on port i. Its only effect on the IS object is to reset the value *interface[i]* to $\perp$, thereby preventing more than one response to an invocation. The *ret_writeread(S)*$_{i,x}$ action

IS_D^{n+1} *object named* IS_x

Signature

Inputs:
$\quad inv\_writeread(v)_{i,x}, \qquad v \in D,\ i \in \{0, \ldots, n\}.$
Internals:
$\quad update(\mathcal{U}), \qquad \mathcal{U} \subseteq \{0, \ldots, n\}.$
Outputs:
$\quad ret\_writeread(S)_{i,x}, \qquad S \in \vartheta(D),\ i \in \{0, \ldots, n\}.$

State

$memory \qquad \in \vartheta(D)$, initially $(\perp, \ldots, \perp)$.
$input\_value \qquad \in \vartheta(D)$, initially $(\perp, \ldots, \perp)$.
$return\_value \qquad \in \vartheta^2(D)$, initially $(\perp, \ldots, \perp)$.
$interface \qquad \in \{inv, ret, \perp\}^{n+1}$, initially $(\perp, \ldots, \perp)$

Transitions

input: *inv_writeread(v)*$_{i,x}$
 Eff: *input_value[i]* $:= v$
 interface[i] $:= inv$

internal: *update*($\mathcal{U}$)
 Pre: $\mathcal{U} \subseteq \{i \mid interface[i] = inv\}$
 Eff: For all i in $\mathcal{U}$ do
  memory[i] := *input_value[i]*
 For all i in $\mathcal{U}$ do
  return_value[i] := *memory*
 For all i in $\mathcal{U}$ do
  interface[i] := *ret*

output: *ret_writeread(S)*$_{i,x}$
 Pre: *interface[i]* $= ret$
 return_value[i] $= S$
 Eff: *interface[i]* $:= \perp$

Figure 2-1: I/O Automaton for an IS_D^{n+1} object with name IS_x.

can only occur after a return value has been written to the i-th cell of the *return_value* array, and the flag *"ret"* has been written to the corresponding cell in the *interface* array.

In the remainder of this section, we will state and prove a few basic properties about IS objects. These properties will be useful later, when we prove the correctness of a topological framework for analyzing the complexity of protocols in models of computation that include multiple IS objects. For the purpose of this discussion, let us for the remainder of this section fix our attention on a single IS_D^{n+1} object, which we will call IS_x. Following Lynch [25], we define executions of the automaton IS_x as follows.

Definition 2.11 *An* execution fragment *of IS_x is either a finite sequence $s_0, a_1, s_1, a_2, s_2, \ldots, a_{r-1}, s_r$ or an infinite sequence $s_0, a_1, s_1, a_2, s_2, \ldots$ of alternating states and actions such that for all k, where $k \geq 1$, the action a_k brings the object from state s_{k-1} to state s_k. An execution fragment beginning with a start state is called an* execution.

We note that the specification of IS_x does not in any way restrict the number of accesses that may occur on any given port in an execution. In fact, it even allows two invocations on a port i before any response is given to the first invocation. In this thesis, however, we will only be using IS objects as building blocks to construct more complicated, long-lived models of computation, not as long-lived objects in their own right. Therefore, we will only consider a restricted class of executions of IS_x, called *one-shot executions*, as defined below.

Definition 2.12 *A* one-shot execution *of IS_x is an execution for which there is at most one invocation and at most one response on each port.*

From Figure 2-1, it is clear that, while *inv_writeread(v)*$_{i,x}$ actions are always enabled, no *ret_writeread(S)*$_{i,x}$ action may occur on a port i before an *inv_writeread(v)*$_{i,x}$ action has occured on the same port. Definition 2.13 provides a convenient notation for matching *inv_writeread(v)*$_{i,x}$ and *ret_writeread(S)*$_{i,x}$ actions in one-shot executions.

Definition 2.13 *For all i, a writeread$(v, S)_{i,x}$* operation *in a one-shot execution α is an inv_writeread(v)$_{i,x}$ action together with the next ret_writeread(S)$_{i,x}$ action on the same port.*

The operation *writeread*$(v, S)_{i,x}$ uses the i-th port of IS_x to write the value v to the i-th cell of *memory* , and subsequently return a snapshot S.

Definition 2.14 *For all i, an* incomplete operation *in a one-shot execution α is an inv_writeread(v)$_{i,x}$ action for which there is no subsequent ret_writeread(S)$_{i,x}$ action on the same port.*

An incomplete operation *inv_writeread(v)$_{i,x}$* in α uses the i-th port of IS_x to write a value v to the i-th cell of *memory*, but there is no subsequent response on the i-th port in α. In the remainder of this section, we will be considering only one-shot executions of IS_x.

Lemma 2.15 *For any two distinct actions update($\mathcal{U}$) and update($\mathcal{U}'$) in a one-shot execution α of IS_x, the index sets $\mathcal{U}$ and $\mathcal{U}'$ are disjoint.*

Proof. Suppose without loss of generality that $\mathcal{U}$ occurs before $\mathcal{U}'$, and suppose $i \in \mathcal{U}$. Immediately after the action *update($\mathcal{U}$)*, *interface[i]* is equal to *"ret"*. Since we are considering a one-shot execution, *interface[i]* will remain equal to *"ret"* for the remainder of the execution. Hence, the precondition of the *update($\mathcal{U}'$)* action guarantees that $i \notin \mathcal{U}'$. □

We can now define what we mean by concurrent operations of IS_x.

Definition 2.16 *Two operations writeread$(v_i, S_i)_{i,x}$ and writeread$(v_j, S_j)_{j,x}$ in a one-shot execution α of IS_x are concurrent if there exists an action update($\mathcal{U}$) in α such that $i, j \in \mathcal{U}$.*

IS_x exhibits a property called *self-containment*: A snapshot returned on port i by a *ret_writeread(S)$_{i,x}$* action in a one-shot execution α must necessarily contain the value written to *memory* in the matching, preceding *inv_writeread(v)$_{i,x}$* action on port i that is part of the same operation. This property is proven in Lemma 2.17.

Lemma 2.17 *Consider any operation writeread$(v_i, S_i)_{i,x}$ in a one-shot execution α of IS_x. Then $S_i[i] \neq \perp$.*

Proof. The input value v_i is written to *memory* by an action *update($\mathcal{U}_i$)*. In the same action, after the v_i has been written to *memory*, the *memory* variable is copied to *return_value[i]*. This value is then returned as a response on port i. It follows that $S_i[i] \neq \perp$ for all i. □

The value returned by a *ret_writeread(S)*$_{i,x}$ action is a 1-dimensional array of type $\vartheta(D)$. In the following lemma, we prove that IS_x exhibits the property that the set of snapshots returned in a one-shot execution can be totally ordered by the prefix relation defined in Definition 2.3.

Lemma 2.18 *Consider any two writeread operations in a one-shot execution* α*, writeread*$(v_i, S_i)_{i,x}$ *and writeread*$(v_j, S_j)_{j,x}$ *. Either* S_i *is a prefix of* S_j*, or* S_j *is a prefix of* S_i*.*

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update*$(\mathcal{U}_i)$ and *update*$(\mathcal{U}_j)$, respectively.

If these actions are the same, that is, if $\mathcal{U}_i = \mathcal{U}_j$, the two operations *writeread*$(v_i, S_i)_{i,x}$ and *writeread*$(v_j, S_j)_{j,x}$ are concurrent. In this case, the value of *memory* that is copied to *return_value[i]* is identical to the value copied to *return_value[j]*, since both are copied by the same *update*$(\mathcal{U}_i)$ action. It follows that $S_i = S_j$. Now suppose $\mathcal{U}_i \neq \mathcal{U}_j$, and suppose *update*$(\mathcal{U}_i)$ occurs after *update*$(\mathcal{U}_j)$. Since no *memory* cells are ever reset, it follows that the *memory* version that is written to *return_value[j]* during *update*$(\mathcal{U}_j)$ is a prefix of the version that is written to *return_value[i]* during *update*$(\mathcal{U}_i)$. Hence S_j is a prefix of S_i. The case where *update*$(\mathcal{U}_j)$ occurs after *update*$(\mathcal{U}_i)$ is similar, and in this case we have that S_i is a prefix of S_j. The lemma follows. □

The next lemma concerns what is referred to as the *immediacy* property of one-shot executions of IS_x. If a value written to *memory* by an invocation on port j is contained in a snapshot on port i, then the snapshot returned on port j is a prefix of that returned on port i. This corresponds to the informal notion of a *writeread* operation on port j happening *before* a writeread operation on port i.

Lemma 2.19 *Consider any two writeread operations in a one-shot execution* α*, writeread*$(v_i, S_i)_{i,x}$ *and writeread*$(v_j, S_j)_{j,x}$*. If* $S_i[j] \neq \perp$ *then* S_j *is a prefix of* S_i*.*

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update*$(\mathcal{U}_i)$ and *update*$(\mathcal{U}_j)$, respectively, and suppose $S_i[j] \neq \perp$. This implies that either v_j was written to *memory* during *update*$(\mathcal{U}_i)$, in which case $\mathcal{U}_i = \mathcal{U}_j$, or the action *update*$(\mathcal{U}_j)$ occured before

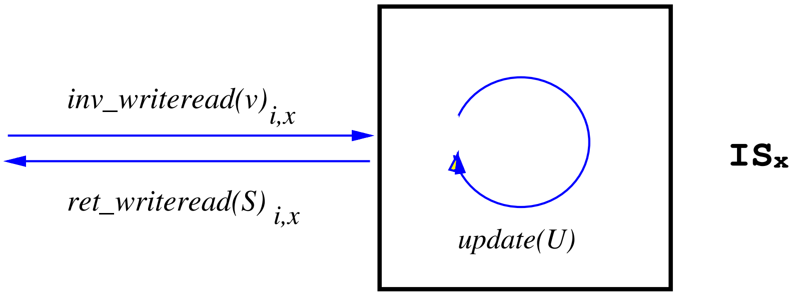

Figure 2-2: Diagram of IS_x.

update$(\mathcal{U}_i)$. In either case, since no *memory* cells are ever reset or written to more than once in a one-shot execution, we have that S_j must be a prefix of S_i. $\square$

Finally, we will prove that one-shot executions are finite, a property that we shall need when discussing fairness in the IIS model, to be introduced in Section 2.4.

Lemma 2.20 *Any one-shot execution α of IS_x is finite.*

Proof. The number of actions of type *inv_writeread*$(v)_{i,x}$ in α is at most $n+1$, one for each port. Similarly, the number of actions of type *ret_writeread*$(S)_{i,x}$ is at most $n+1$. By Lemma 2.15, the number of *update*$(\mathcal{U})$ actions is bounded by $n+1$ as well. Hence α is finite. $\square$

Figure 2-2 shows a stylized diagram of the IS object IS_x.

2.4 The Iterated Immediate Snapshot (IIS) Model

The *iterated immediate snapshot (IIS) model*, first used implicitly by Herlihy and Shavit [22, 26], was recently formulated as a computation model by Borowsky and Gafni [10]. The model assumes a finite sequence $IS_D^{n+1}, IS_{\vartheta(D)}^{n+1}, IS_{\vartheta^2(D)}^{n+1}, \ldots, IS_{\vartheta^{k-1}(D)}$ of IS objects, denoted by $IS_1, IS_2, IS_3, \ldots, IS_k$, where $k > 0$. In the *k-shot IIS model*, there are k available IS objects in sequence.

Each protocol in the IIS model is fully determined by the maximum number $n+1$ of processes that can participate, the number k of IS objects available, and a decision map

Environment E

Signature

Inputs:
  decide(S)$_i$, $S \in \vartheta^k(D)$.
Internals:

Outputs:
  start(v)$_i$, $v \in D$.
  fail$_i$

State

status $\in \{\perp, participating, started, decided, failed\}^{n+1}$, initially in $\{\perp, participating\}^{n+1}$.

Transitions

input: *decide(S)*$_i$
  Eff: *status* := *decided*

output: *fail*$_i$
  Pre: *status* = *started*
  Eff: *status* := *failed*

output: *start(v)*$_i$
  Pre: *status* = *participating*
  Eff: *status* = *started*

Figure 2-3: I/O Automaton for environment E in $\mathcal{P}_{(n,k,\delta)}$.

$\delta : \vartheta^k(D) \to D_O$, where D_O is an arbitrary data type, which we call the protocol's *output data type*. We refer to the protocol obtained by fixing these parameters as $\mathcal{P}_{(n,k,\delta)}$.

We can specify the $\mathcal{P}_{(n,k,\delta)}$ protocol using I/O automata [24, 25] as follows. Each IS object is specified as in figure 2-1, and each process i is specified as in Figure 2-4. The external environment E is modeled as an automaton as specified in Figure 2-3. The protocol $\mathcal{P}_{(n,k,\delta)}$ can then be specified by composing the automata for the processes with the Environment automaton and the IS object automata by matching up operations as follows. Assume that for each process i, and each x, where $1 \leq x \leq k$, the object IS_x is connected to the x-th port on i, and the i-th process is connected to the i-th port on IS_x. For each processor i and IS object IS_x, we match up i's and IS_x's *inv_writeread(v)*$_{i,x}$ and *ret_writeread(S)*$_{i,x}$ actions. Moreover, each process i's *start(v)*$_i$, *decide(S)*$_i$ and *fail*$_i$ actions are matched up with E's *start(v)*$_i$, *decide(S)*$_i$ and *fail*$_i$ actions. The resulting protocol automaton is denoted $\mathcal{P}_{(n,k,\delta)} = \{E; 0, 1, \ldots, n; IS_1, IS_2, \ldots, IS_k\}$.

In any execution α of $\mathcal{P}_{(n,k,\delta)}$, the first action executed by any process i is a *start(v)*$_i$ action, which stores an *input value* v in its *local_state* variable. Lemma 2.21 states that in a given execution α, some processes may not receive a *start(v)*$_i$ action, and duplicate *start(v)*$_i$ actions may not occur.

Lemma 2.21 *In any execution α of $\mathcal{P}_{(n,k,\mu)}$, the environment E may issue at most one start(v)$_i$ action for each process i.*

Proof. The proof follows immediately from the specification of the environment E in Figure 2-3. A *start(v)$_i$* action occurs only if the *status[i]* field is initially set to *participating*. Moreover, since the *status[i]* field is set to *started* by any *start(v)$_i$* action, and never reset to *participating*, the precondition of *start(v)$_i$* prevents duplicate *start(v)$_i$* actions. □

In the case where a process i does not have a *start(v)$_i$* action in α, i takes no steps at all in this particular execution.

Definition 2.22 *A process i is said to* participate *in an execution α of the protocol $\mathcal{P}_{(n,k,\delta)}$ if α contains a start(v)$_i$ action. The set of participating processes in α is called the* participating set *in α.*

A process i is said to *decide* in an execution α when it executes a *decide(S)$_i$* action. After executing a *decide(S)$_i$* action, a process i does not take any further steps in α. The value S returned by the *decide(S)$_i$* action is process i's *output value* in α.

For any execution α of $\mathcal{P}_{(n,k,\delta)}$, the processes' input values can conveniently be represented using an $n+1$-dimensional *input vector* $\vec{I}$, as specified in the previous section, with input data type D. The i-th entry of $\vec{I}$ is the input of process i. Similarly, the processes' output values in α can be represented using an $n+1$-dimensional *output vector* $\vec{O}$. The i-th entry of $\vec{O}$ is the output of process i.

It should be noted that the notions of participating processes and sets defined for executions and input vectors are consistent; A process i participates in an execution α if and only if the index i participates in the input vector $\vec{I}$ corresponding to α. Therefore, when the meaning is clear from the context, we usually omit qualifying a participating set with an execution or input vector.

Suppose for now that no processes fail in the execution α of $\mathcal{P}_{(n,k,\delta)}$, that is, for all i, no *fail$_i$* action occurs in α. We will discuss the issue of failures later in this section. In the non-failing case, each participating process i starts by accessing the first IS object by executing a *writeread(v)$_{i,IS_1}$* action, where v is equal to *local_state*, which initially contains the input of process i. The response consists of a snapshot $S_i \in \vartheta(D)$ of the first IS object's shared memory vector, containing the input values of some of the participating processes. In particular, process i's input is always contained in the output i receives. Upon receiving the response S_i, process i copies this value to its *local_state* variable, after which it accesses the second IS object by executing a $writeread(v)_{i,IS_2}$ action, where v is equal to *local_state*. In effect, the output from the first IS object is used as an input to the second. Each non-failing, participating process accesses *exactly* k IS objects in sequence in this manner, before deciding. The number k of IS objects accessed is the same for each non-failing, participating process in any execution of the protocol. Once a process has received an output from the k-th IS object and stored this value in the *local_state* variable, it applies a decision map δ to this value, which maps the local state to an output value, stores the result in *local_state*, and halts. The *local_state* variable now contains the process' decision value.

All executions of any protocol $\mathcal{P}_{(n,k,\delta)}$ are necessarily finite, as stated in Lemma 2.23.

Lemma 2.23 *Let α be an execution of the protocol $\mathcal{P}_{(n,k,\delta)}$. Then α is finite.*

Proof. It follows from Lemma 2.21 that α contains at most $n+1$ actions of type $start(v)_i$. It is also clear from the specification of E that it contains at most $n+1$ actions of type $decide(S)_i$ and at most $n+1$ actions of type $fail_i$. Now, for all x, where $1 \leq x \leq k$, the restriction of α to IS_x, denoted α_x, is a one-shot execution of IS_x. It follows that α_x is finite by Lemma 2.20. Now, every action of α necessarily belongs to the restriction α_x for some x, where $1 \leq x \leq k$, and so we conclude that α is finite. □

We are particularly interested in protocols that guarantee strong fault-tolerance conditions. The failure model we shall be concerned with only involves stopping failures, that is, we assume that each process i may simply stop without warning, after which it issues no further locally controlled actions. We model this using the $fail_i$ action, as shown in Figure 2-4.

Definition 2.24 *A process i* fails *in an execution α of $\mathcal{P}_{(n,k,\delta)}$ if α contains a $fail_i$ action.*

To ensure that the $fail_i$ action achieves the stopping failure property we want, we need the following lemma.

Process i

Signature

Inputs:

$ret\_writeread(S)_{i,x}$, $S \in \bigcup_{l=1}^{k} \vartheta^l(D)$.
$start(v)_i$, $v \in D$.
$fail_i$

Internals:

Outputs:

$inv\_writeread(v)_{i,x}$, $v \in \bigcup_{l=0}^{k} \vartheta^l(D)$.
$decide(S)_i$, $S \in \vartheta^k(D)$.

State

$local\_state$ $\in (\bigcup_{l=0}^{k} \vartheta^l(D)) \cup \{\perp\}$, initially in the fresh state $\perp$.
$counter$ $\in \mathbb{Z}$, initially 1.
$status$ $\in \{ready, waiting, decided, failed\}$, initially *ready*.

Transitions

input: $ret\_writeread(S)_{i,x}$
Eff: $local\_state := S$
$counter := counter + 1$
If *status waiting* do
$status := ready$

output: $inv\_writeread(v)_{i,x}$
Pre: $counter = x \leq k$
$local\_state = v \neq \perp$
$status = ready$
Eff: $status := waiting$

input: $fail_i$
Eff: $status := failed$

input: $start(v)_i$
Eff: $local\_state := v$

output: $decide(S)_i$
Pre: $counter > k$
$\mu(local\_state) = S$
$status = ready$
Eff: $status := decided$

Figure 2-4: I/O Automaton for process i running k-shot IIS protocol.

Lemma 2.25 *Let α be an execution of $\mathcal{P}_{(n,k,\delta)}$ that contains a $fail_i$ action. Then α contains no actions locally controlled by i ($inv\_writeread(v)_{i,x}$ or $decide(S)_i$) after the $fail_i$ action.*

Proof. The $fail_i$ action sets the *status* variable to the value *failed*. Since all actions locally controlled by i require the *status* variable to have the value *ready*, and no input actions may change the *status* variable once it has been set to *failed*, the lemma follows. □

We do not consider other, more complicated models, such as the Byzantine failure model, in which processes may exhibit arbitrary behavior. We use the following fairness condition, adapted from Lynch [25], for protocols in the IIS model.

Definition 2.26 *An execution α of a protocol $\mathcal{P}_{(n,k,\delta)}$ is* fair *if, for all i, in the final state of α, no locally controlled action of process i is enabled.*

In the remainder of this thesis, unless stated otherwise, we restrict our attention to the set of fair executions of a protocol. Furthermore, we will focus on protocols with the property that any non-failing process eventually decides, independent of the progress of the other processes. This is called *wait-free* termination. In this thesis, we use the following formalization of the wait-free termination condition, also adapted from Lynch [25].

Definition 2.27 *A protocol $\mathcal{P}_{(n,k,\delta)}$ is* wait-free *if, in any fair execution α of $\mathcal{P}_{(n,k,\delta)}$, for all i, and all x, where $1 \leq x \leq k$, every $inv\_writeread(v)_{i,x}$ action issued by a process i has a following $ret\_writeread(S)_{i,x}$ action.*

The IIS protocols we consider trivially satisfy the wait-free termination condition, since, for all x, y, where $1 \leq x, y \leq k$, invocations and responses on any given port i of IS_x are independent of invocations and responses on any other port j of IS_y (note that x and y may be equal).

A stylized interconnection diagram of the k-shot IIS model is given in Figure 2-5. The diagram does not show any internal actions, nor does it show the *start*, *decide* and *fail* actions of the processes.

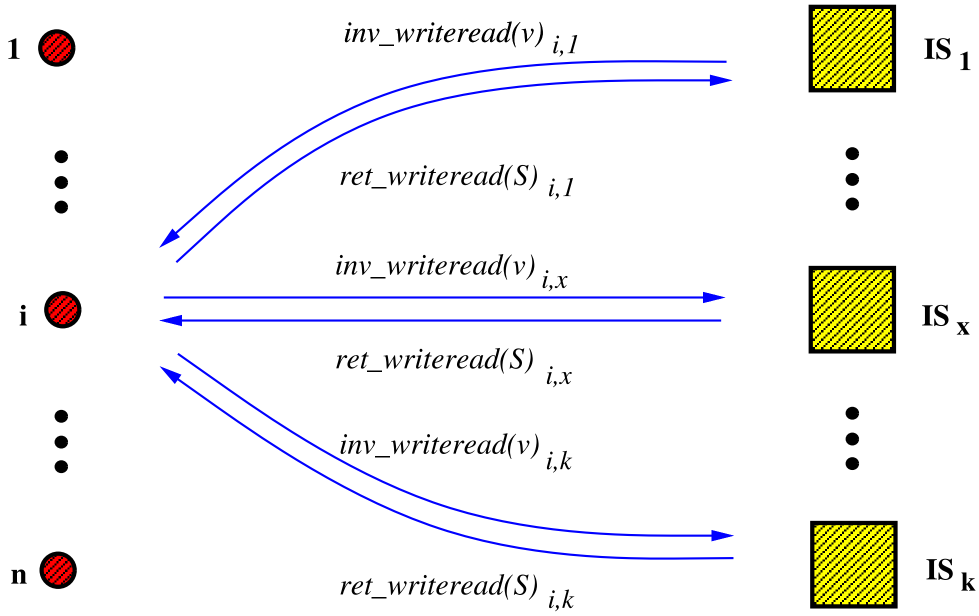

Figure 2-5: Diagram of the k-shot IIS model.

2.5 The Non-Uniform Iterated Immediate Snapshot (NIIS) Model

In this section, we generalize the IIS model by introducing the *non-uniform IIS (NIIS) model.* In the standard IIS model, the number of IS objects available is finite, and the number of IS objects accessed is fixed over all processes and all executions. The NIIS model, however, assumes an infinite sequence $IS^{n+1}_D, IS^{n+1}_{\vartheta(D)}, IS^{n+1}_{\vartheta^2(D)}, \ldots$ of IS objects, denoted $IS_1, IS_2, IS_3, \ldots$. The number of IS objects accessed by any two distinct processes in a given execution need not be the same, and, moreover, the number of objects accessed by any fixed process may vary from execution to execution. The motivation behind this is to be able to model complexity more accurately, as will be shown in later sections.

Each protocol in the NIIS model can be fully characterized by the maximum number $n+1$ of processes that can participate, a predicate function $\tau : \bigcup_{l=0}^{k} \vartheta^l(D) \to \{\texttt{true}, \texttt{false}\}$, which each process applies to its *local_state* variable after each complete *writeread* operation to determine whether or not to decide, and a decision map $\delta : \bigcup_{l=0}^{k} \vartheta^l(D) \to D_O$, where D_O is an arbitrary data type, which we call the protocol's *output data type*. We refer to the protocol obtained by fixing these parameters as $\mathcal{P}_{(n,\tau,\delta)}$.

We can specify the $\mathcal{P}_{(n,\tau,\delta)}$ protocol using I/O automata [24, 25] as follows. Each IS ob-

Environment E

Signature

Inputs:
decide(S)$_i$, $S \in \bigcup_{l \geq 1} \vartheta^l(D)$.
Internals:

Outputs:
start(v)$_i$, $v \in D$.
fail$_i$

State

status $\in \{\perp, participating, started, decided, failed\}^{n+1}$, initially in $\{\perp, participating\}^{n+1}$.

Transitions

input: *decide(S)$_i$*
Pre: *status* $=$ *started*
Eff: *status* $:=$ *decided*

output: *start(v)$_i$*
Pre: *status* $=$ *participating*
Eff: *status* $=$ *started*

output: *fail$_i$*
Pre: *status* $=$ *started*
Eff: *status* $:=$ *failed*

Figure 2-6: I/O Automaton for environment E in $\mathcal{P}_{(n,\tau,\delta)}$.

ject is specified as in Figure 2-1, and each process i is specified as in Figure 2-7. The external environment E is modeled as an automaton as specified in Figure 2-6. The protocol $\mathcal{P}_{(n,\tau,\mu)}$ can then be specified by composing the automata for the processes with the environment E automaton and the IS object automata by matching up operations as follows. Assume that for each process i, the x-th IS object, IS_x, is connected to the x-th port on i, and for each IS object IS_x, the i-th process is connected to the i-th port on IS_x. For each processor i and IS object IS_x, we match up i's and IS_x's *inv_writeread(v)$_{i,x}$* and *ret_writeread(S)$_{i,x}$* actions. Moreover, each process i's *start(v)$_i$* , *decide(S)$_i$* and *fail$_i$* actions are matched up with E's *start(v)$_i$* , *decide(S)$_i$* and *fail$_i$* actions. The resulting protocol automaton is denoted $\mathcal{P}_{(n,\tau,\delta)} = \{E; 0, 1, \ldots, n; IS_1, IS_2, \ldots\}$.

The only significant difference between a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model and a protocol in the IIS model is that after each complete *writeread* operation, each process checks whether it has reached a final state by applying the predicate τ to the *local_state* variable. If τ returns `true`, the process executes a *decide(S)$_i$* action and halts. Otherwise, it accesses the next IS object as in the IIS model, and so on. In fact, any protocol $\mathcal{P}_{(n,k,\delta)}$ in the IIS model is equivalent to a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, in which the predicate τ simply checks whether or not the *local_state* variable is of type $\vartheta^k(D)$ or not.

Notice that, unlike the IIS model, the NIIS model permits infinite executions, for some choices of the termination predicate map τ. We only consider protocols for which τ is chosen such that the entire system does not have any infinite executions, however. The failure model we use for the NIIS model is the same as for the IIS model, that is, we allow stopping failures, in which a process simply stops taking steps. As for the IIS model, we model stopping failures using $fail_i$ actions, as shown in Figure 2-7.

We use the following fairness condition, adapted from Lynch [25], for protocols in the NIIS model.

Definition 2.28 *An execution α of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ is* fair *if α is finite, and in the final state no locally controlled action of process i is enabled.*

In the remainder of this thesis, unless stated otherwise, we restrict our attention to the set of fair executions of a protocol. We use the following *wait-free* termination condition, also adapted from Lynch [25], for protocols in the NIIS model.

Definition 2.29 *A protocol $\mathcal{P}_{(n,\tau,\delta)}$ is* wait-free *if, in any fair execution α of $\mathcal{P}_{(n,\tau,\delta)}$, for all i, and all x, where $1 \leq x$, every inv_writeread$(v)_{i,x}$ action issued by a process i is followed by a ret_writeread$(S)_{i,x}$ action.*

The NIIS protocols we consider trivially satisfy the wait-free termination condition, since, for all x, y, where $1 \leq x, y \leq k$, invocations and responses on any given port i of IS_x are independent of invocations and responses on any other port j of IS_y (note that x and y may be equal).

A stylized interconnection diagram of the NIIS model is given in Figure 2-8. The diagram does not show any internal actions, nor does it show the *start* and *decide* actions of the processes.

2.6 Solvability of Decision Tasks

In this section we specify the safety property [25] we will consider in this thesis, which we call *solvability*. We will define precisely what it means to solve a decision task in the NIIS

Process i

Signature

Inputs:

$ret\_writeread(S)_{i,x}$, $\quad S \in \bigcup_{l=1}^{k} \vartheta^l(D)$.
$start(v)_i$, $\quad v \in D$.
$fail_i$

Internals:

Outputs:

$inv\_writeread(v)_{i,x}$, $\quad v \in \bigcup_{l=0}^{k} \vartheta^l(D)$.
$decide(S)_i$, $\quad S \in \bigcup_{l=0}^{k} \vartheta^l(D)$.

State

$local\_state \in (\bigcup_{l=0}^{k} \vartheta^l(D)) \cup \{\perp\}$, initially in the fresh state $\perp$.
$status \in \{ready, waiting, decided, failed\}$, initially *ready*.

Transitions

input: $ret\_writeread(S)_{i,x}$
Eff: $local\_state := S$
If *status waiting* do
$status := ready$

output: $inv\_writeread(v)_{i,x}$
Pre: $\tau(local\_state) =$ `false`
$status = ready$
$local\_state = v \neq \perp$
Eff: $status := waiting$

input: $fail_i$
Eff: $status := failed$

input: $start(v)_i$
Eff: $local\_state := v$

output: $decide(S)_i$
Pre: $\tau(local\_state) =$ `true`
$\mu(local\_state) = S$
$status = ready$
Eff: $status := decided$

Figure 2-7: I/O Automaton for process i running NIIS protocol.

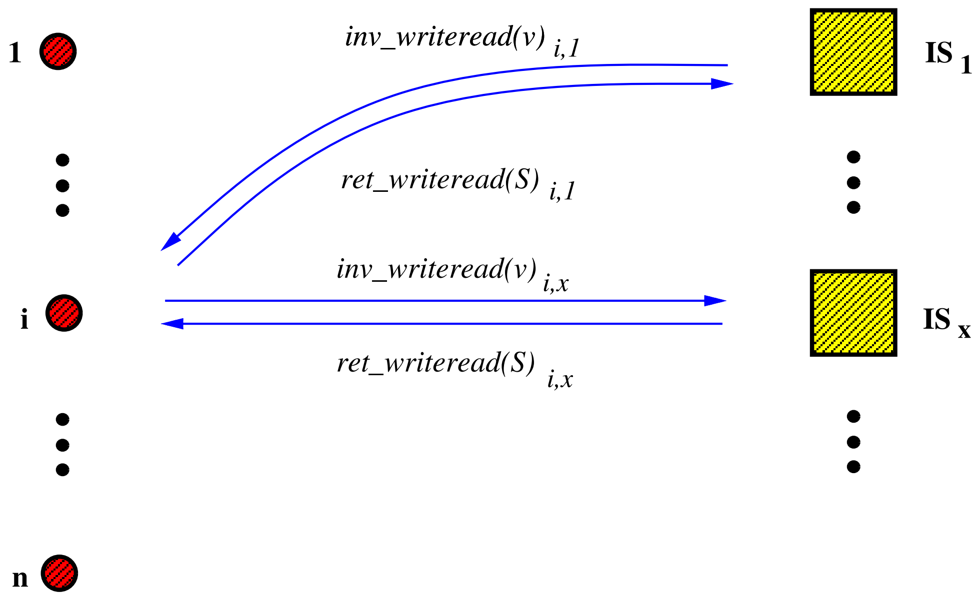

Figure 2-8: Diagram of the NIIS model.

model. Since the IIS model is equivalent to a special case of the NIIS model, the definitions below also apply directly to the IIS model. Although we are primarily interested in the fair executions of a given protocol, these definitions do hold for all executions of a protocol.

Definition 2.30 *Suppose we are given a decision task $\mathcal{D} = \langle I, O, \gamma \rangle$, and an execution α of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, with corresponding input and output vectors $\vec{I}$ and $\vec{O}$, respectively. Then $\mathcal{P}_{(n,\tau,\delta)}$ solves $\mathcal{D}$ in α if $\vec{O}$ is a prefix of some vector in $\gamma(\vec{I})$.*

Informally, this means that if $\mathcal{P}_{(n,\tau,\delta)}$ solves a task $\mathcal{D} = \langle I, O, \gamma \rangle$ in α, then the outputs of α are consistent with some continuation α' of α, where the outputs of α' correspond to some vector in $\gamma(\vec{I})$.

Definition 2.31 *A protocol $\mathcal{P}_{(n,\tau,\delta)}$* solves *a decision task $\mathcal{D} = \langle I, O, \gamma \rangle$ if it solves it in every execution α.*

2.7 Complexity Measures for the IIS and NIIS Models

We now define the complexity measures to be used for analyzing the performance of protocols in the NIIS model. Since the IIS model is equivalent to a special case of the NIIS model, these measures also apply directly to the IIS model. As we will see however, these measures expose a potential weakness of the IIS model: All processes always take the same number of steps in every execution of a given protocol.

Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol in the NIIS model solving a given decision task $\mathcal{D}$, let $\vec{I}$ be an input vector, and let α be any execution of $\mathcal{P}_{(n,\tau,\delta)}$ that corresponds to $\vec{I}$. For all i, let t_i be the number of IS objects accessed by process i in α. Clearly, t_i is an integer. We first define the time complexity of the execution α.

Definition 2.32 *The time complexity of α, denoted t_α, is $\max_i t_i$, the maximum number of IS objects accessed by any process.*

We note that t_α is well-defined, since the number of processes $n+1$ is finite. Moreover, by definition of the max function, t_α is an integer value. We use the definition given above to define the time complexity of the protocol $\mathcal{P}$ on the input vector $\vec{I}$.

Definition 2.33 *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on $\vec{I}$, denoted $t_{\vec{I}}$, is the supremum of the set $\{t_\alpha \mid t_\alpha$ is an execution corresponding to $\vec{I}\}$.*

Finally, we define the complexity of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ on an input set I.

Definition 2.34 *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on I, denoted t_I, is the supremum of the set $\{t_{\vec{I}} \mid \vec{I} \in I\}$.*

The reason for preferring these simple, discrete complexity measures over other, more elaborate measures such as real time, for instance, is the highly regular structure of the IIS and NIIS models. We make the assumption that each access to an IS object takes the same amount of time, and do not worry about breaking up the time required to complete each access to an IS object into subparts. Instead, we group the time spent on invocation, response, and on local computation at the IS object. This assumption is somewhat strong, as the presence of asynchrony in our model will tend to introduce varying delays for each access to an object. However, we believe that, as a first step towards a complexity theory, this assumption is justifiable, as it allows for complexity measures that are simple and easy to apply, and that have a particularly nice topological representation, as we will see in Chapter 3.

The motivation behind introducing non-uniformity to the IIS model is to give a more accurate description of the complexity of solving decision tasks. With our complexity measures, the required amount of computation in order to solve a given decision task in the NIIS model may vary from process to process, from input value to input value, and indeed from execution to execution. Were we to restrict our attention to the uniform IIS model, however, the complexity measures defined above would be somewhat less useful, since in this model, all processes always take the same number of steps in a given execution, and any process will take the same number of steps in all executions in which it participates. In other words, the complexity of solving a task in the IIS model is always the same across all processes and all executions, namely the number k of IS objects available. Thus, the introduction of non-uniformity allows us to more accurately capture the complexity of solving decision tasks.

Chapter 3

Topological Framework

In this section we first introduce some known tools from the field of algebraic topology and show how they may be used to model decision tasks and protocols in the NIIS model of computation. We then introduce a new tool for analyzing complexity in this setting, called the non-uniform iterated chromatic subdivision.

3.1 Basic Topological Definitions and Concepts

This section introduces the basic topological definitions and concepts that we shall need for modeling decision tasks and wait-free protocols in the NIIS model. Some of these definitions are fairly standard, and are mainly taken from popular textbooks on algebraic topology [27, 31], while others are due to Herlihy and Shavit [21, 22, 26]. Some of the figures used in this section are also adopted from Herlihy and Shavit's work [21, 22, 26].

A *vertex* $\vec{v}$ is a point in a Euclidian space R^l. A set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of vertexes is *geometrically independent* if and only if the set of vectors $\{\vec{v}_i - \vec{v}_0\}_{i=1}^{n}$ is linearly independent. Clearly, for a set of $n+1$ vertexes to be geometrically independent, $l \geq n$. We can now define the concept of a geometric simplex, or simplex for short.

Definition 3.1 *Let $\{\vec{v}_0, \ldots, \vec{v}_n\}$ be a geometrically independent set of vertexes in R^l. We define the n-simplex S spanned by $\vec{v}_0, \ldots, \vec{v}_n$ to be the set of all points x such that $x = \sum_{i=0}^{n} t_i \vec{v}_i$ where $\sum_{i=0}^{n} t_i = 1$ and $t_i \geq 0$ for all i.*

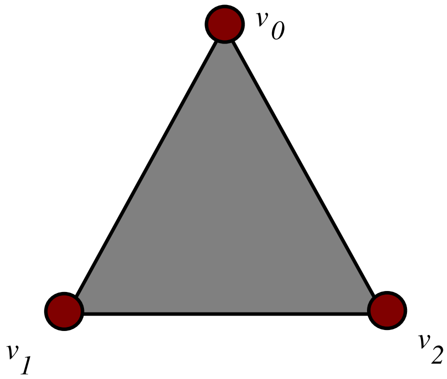

Figure 3-1: Example of a 2-simplex.

For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. For an example of a 2-simplex see Figure 3-1. For simplicity, we often denote the simplex spanned by a set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of geometrically independent vertexes as $(\vec{v}_0, \ldots, \vec{v}_n)$. The number n is called the *dimension* of the simplex S, and is often denoted by $\dim(S)$. For clarity, we will sometimes include the number n as an explicit superscript when referring to a simplex, that is, we will write S^n to refer to the simplex spanned by the vertexes in $\{\vec{v}_0, \ldots, \vec{v}_n\}$.

Any simplex T spanned by a subset of $\{\vec{v}_0, \ldots, \vec{v}_n\}$ is called a *face* of S. The faces of S different from S itself are called the *proper* faces of S. The simplex spanned by the vertexes $\{\vec{v}_0, \vec{v}_1\}$ is a proper face of the 2-simplex in Figure 3-1.

The union of the proper faces of S is called the *boundary* of S, and is denoted $Bd(S)$. The interior of S, denoted $Int(S)$, is defined by the set equation $Int(S) = S - Bd(S)$.

We will use a vertex to model the state of a single process, and a simplex to model consistent states of all the processes involved in solving a decision task or in running a protocol in the NIIS model. To model a collection of such states we need the concept of a geometric, simplicial complex, or complex for short, which is defined below.

Definition 3.2 *A geometric simplicial complex $\mathcal{K}$ in the Euclidean space R^l is a collection of geometric simplexes in R^l such that*

- *Every face T of every simplex S in $\mathcal{K}$, where* $\dim(T) \leq \dim(S)$, *is contained in $\mathcal{K}$.*

- *The intersection U of any two simplexes S, T in K, where* $\dim(U) \leq \dim(S), \dim(T)$, *is contained in $\mathcal{K}$.*

In this thesis we will only consider finite complexes. The *dimension* of a complex $\mathcal{K}$, often denoted by $\dim(\mathcal{K})$, is the highest dimension of any of its simplexes, and is also sometimes indicated explicitly by a superscript. An n-dimensional complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. All complexes considered in this paper are pure, unless stated otherwise. A simplex S in $\mathcal{K}$ with dimension $\dim(S) = \dim(\mathcal{K})$ is called a *maximal* simplex.

Given a simplex S, let $\mathcal{S}$ denote the complex of all faces of S, and let $\dot{\mathcal{S}}$ denote the complex consisting of all proper faces of S. We note that, since $\dot{\mathcal{S}}$ contains all faces of S except S itself, $\dim(\dot{\mathcal{S}}) = \dim(S) - 1$. An example of a pure, 2-dimensional simplicial complex, which we call $\mathcal{K}$, is shown in Figure 3-2. This complex equals the union of $\dot{\mathcal{S}}$ and $\dot{\mathcal{T}}$, where S is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_2\}$, and T is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_3\}$. Both S and T are maximal simplexes in this example.

If $\mathcal{L}$ is a subcollection of $\mathcal{K}$ that is closed under containment and intersection, where $\dim(\mathcal{L}) \leq \dim(\mathcal{K})$, then $\mathcal{L}$ is a complex in its own right. It is called a *subcomplex* of $\mathcal{K}$. For example, the complex $\dot{\mathcal{S}}$ of faces of S is a subcomplex of $\mathcal{K}$ in Figure 3-2

One subcomplex of a complex $\mathcal{K}$ of particular interest is the subcomplex of all simplexes in $\mathcal{K}$ of dimension at most p, where p is some integer between 0 and $\dim(\mathcal{K})$. We call this subcomplex the p-th *skeleton* of a $\mathcal{K}$, denoted $skel^p(\mathcal{K})$. The elements of the collection $skel^0(\mathcal{K})$ are called the *vertexes* of $\mathcal{K}$. The 0-skeleton of the complex $\mathcal{K}$ in Figure 3-2 is the collection of vertexes $\{\{\vec{v}_0\}, \ldots, \{\vec{v}_3\}\}$. Similarly, the 1-skeleton of $\mathcal{K}$ is the union of the 0-skeleton described above and the collection $\{\{\vec{v}_0, \vec{v}_1\}, \ldots \{\vec{v}_0, \vec{v}_3\}, \{\vec{v}_1, \vec{v}_2\}, \{\vec{v}_1, \vec{v}_3\}\}$.

We now define a way of "adding" simplexes, known as *starring.*

Definition 3.3 *Let $S = (s_0, \ldots, s_p)$ and $T = (t_0, \ldots, t_q)$ be simplexes. Then the* star *of S and T, denoted $S \star T$ is the simplex $(s_0, \ldots, s_p, t_0, \ldots, t_q)$.*

We may extend the notion of starring to complexes as well, as shown below.

Definition 3.4 *Let $\mathcal{K}$ and $\mathcal{L}$ be simplicial complexes, not necessarily of the same dimension. Then the* star *of $\mathcal{K}$ and $\mathcal{L}$, denoted $\mathcal{K} \star \mathcal{L}$, is the collection of simplexes $\mathcal{K} \cup \mathcal{L} \cup \{S \star T \mid S \in \mathcal{K}, T \in \mathcal{L}\}$.*

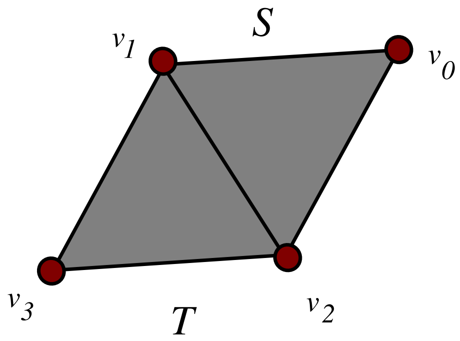

Figure 3-2: Example of a pure, 2-dimensional simplicial complex.

The star of two complexes $\mathcal{K}$ and $\mathcal{L}$ is a complex in its own right [27].

Let $|K|$ be the subset $\bigcup_{S\in\mathcal{K}} S$ of R^l that is the union of the simplexes of $\mathcal{K}$. Giving each simplex its natural topology as a subspace of R^l, we topologize $|\mathcal{K}|$ by declaring a subset A of $|\mathcal{K}|$ to be closed iff $A \cap S$ is closed for all $S \in \mathcal{K}$. This space is called the *polytope* of $\mathcal{K}$. Conversely, $\mathcal{K}$ is called a *triangulation* of $|\mathcal{K}|$.

In practice, the geometric representations we have given for simplexes and complexes are not always convenient, since the analytic geometry involved can get quite involved. Therefore, we introduce the notions of *abstract simplexes* and *abstract complexes.*

Definition 3.5 *An abstract simplex S is a finite, nonempty set.*

The *dimension* of S is its cardinality. Each nonempty subset T of S is called a *face* of S. Each element of S is called a *vertex* of S. There is a close relationship between geometric simplexes and abstract simplexes; Any geometrically independent set of vectors $\{\vec{v}_0, \dots, \vec{v}_n\}$ not only span a geometric simplex, they also form an abstract simplex.

Definition 3.6 *An abstract complex $\mathcal{K}_a$ is a collection of abstract simplexes, such that if S is in $\mathcal{K}_a$, so is any face of S.*

Most concepts defined for geometric complexes immediately carry over to abstract complexes; The *dimension* of $\mathcal{K}_a$, often denoted by $\dim(\mathcal{K}_a)$, is the highest dimension of any of its simplexes. An n-dimensional abstract complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. If $\mathcal{L}_a$ is a subcollection of $\mathcal{K}_a$ that is itself an abstract complex, then $\mathcal{L}_a$ is called a *subcomplex* of $\mathcal{K}_a$.

Definition 3.7 *Let $\mathcal{K}$ be a geometric complex, and let V be the vertex set of $\mathcal{K}$. Let $\mathcal{K}_a$ be the collection of all subsets S of V such that S spans a simplex in $\mathcal{K}$. Then $\mathcal{K}_a$ is called the* vertex scheme *of $\mathcal{K}$.*

Definition 3.8 *Two abstract complexes $\mathcal{K}_a$ and $\mathcal{L}_a$ are* isomorphic *if there is a bijective correspondence ψ between their vertex sets such that a set S of vertexes is in $\mathcal{K}_a$ iff $\psi(S) \in \mathcal{L}_a$. The bijective correspondence ψ is called an* isomorphism.

Theorem 3.9 *Every abstract complex $\mathcal{K}_a$ is isomorphic to the vertex scheme of some geometric complex $\mathcal{K}$ in $\mathbb{R}^{2\dim(\mathcal{K}_a)+1}$.*

We will not prove this theorem here. For a proof, see any standard textbook on algebraic topology [27, 31]. In the rest of this thesis, for convenience, we will often use abstract and geometric representations of simplexes and complexes interchangeably. The remainder of this section, however, which introduces a number of important topological concepts, such as simplicial maps, subdivisions and carriers, is set in the context of geometric complexes.

We first define the notions of simplicial vertex maps and simplicial maps from one complex into another.

Definition 3.10 *Let $\mathcal{K}$ and $\mathcal{L}$ be complexes, possibly of different dimensions, and let $\mu : skel^0(\mathcal{K}) \to skel^0(\mathcal{L})$ be a function mapping vertexes to vertexes. Suppose that whenever the vertexes $\vec{v}_0, \ldots, \vec{v}_n$ of $\mathcal{K}$ span a simplex of $\mathcal{K}$, the vertexes $\mu(\vec{v}_0), \ldots, \mu(\vec{v}_n)$ span a simplex of $\mathcal{L}$. Then μ is called a simplicial vertex map from $\mathcal{K}$ to $\mathcal{L}$. μ can be extended to a continuous map $\mu_* : |\mathcal{K}| \to |\mathcal{L}|$ such that*

$$x = \sum_{i=0}^{n} t_i \vec{v}_i \Rightarrow \mu_*(x) = \sum_{i=0}^{n} t_i \mu_*(\vec{v}_i)$$

This continuous extension is called a simplicial map from $\mathcal{K}$ to $\mathcal{L}$.

For simplicity, we henceforth refer to the simplicial vertex map μ as the simplicial map, without actual reference to the continuous extension μ_*, which is less relevant for our

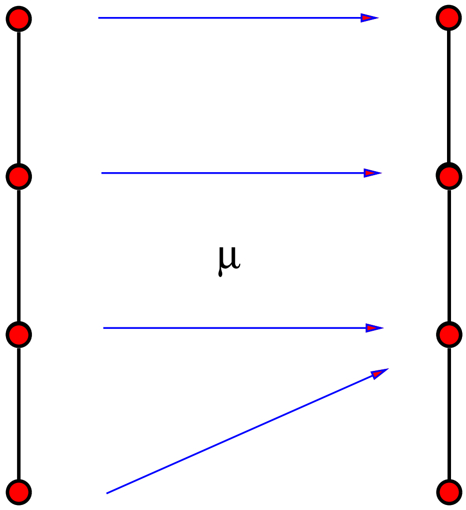

Figure 3-3: Example of a simplicial map between two complexes.

purposes. As a further abuse of notation, we usually write $\mu : \mathcal{K} \rightarrow \mathcal{L}$ when we refer to the simplicial vertex map, glossing over the fact that this map is in fact only defined on the vertexes of $\mathcal{K}$, and that the image of the map is a subset of the vertex set of $\mathcal{L}$. Henceforth, unless stated otherwise, all maps between complexes are assumed to be simplicial. An example of a simplicial map is given in Figure 3-3.

We note that a simplex and its image under a simplicial map need not have the same dimension. A simplicial map $\mu : \mathcal{K} \rightarrow \mathcal{L}$ is *non-collapsing* if it preserves dimension, that is, for all $S \in \mathcal{K}$: $\dim(\mu(S)) = \dim(S)$.

Definition 3.11 *A* coloring *of an n-dimensional complex $\mathcal{K}$ is a non-collapsing simplicial map $\chi : \mathcal{K} \rightarrow \mathcal{S}$, where S is an n-simplex.*

Intuitively, a coloring corresponds to a labeling of the vertexes of the complex such that no two neighboring vertexes (connected by a 1-simplex) have the same label. A *chromatic complex* $(\mathcal{K}, \chi)$ is a complex $\mathcal{K}$ together with a coloring χ of $\mathcal{K}$. An example of a chromatic complex is given in Figure 3-4, where the colors are the values $\{1, 2, 3\}$. When it is clear from the context, we specify the chromatic complex $(\mathcal{K}, \chi)$ simply as the complex $\mathcal{K}$, omitting explicit mention of the coloring χ.

Definition 3.12 *Let $(\mathcal{K}, \chi_{\mathcal{K}})$ and $(\mathcal{L}, \chi_{\mathcal{L}})$ be chromatic complexes, and let $\mu : \mathcal{K} \rightarrow \mathcal{L}$ be a simplicial map. We say that μ is* chromatic *if, for every vertex $\vec{v} \in \mathcal{K}$, $\chi_{\mathcal{K}}(\vec{v}) = \chi_{\mathcal{L}}(\mu(\vec{v}))$.*

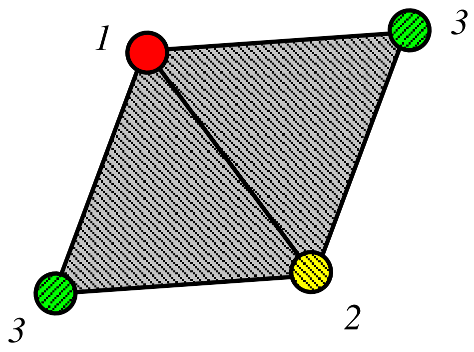

Figure 3-4: Example of a 1-dimensional chromatic complex.

In other words, μ is chromatic if it maps each vertex in $\mathcal{K}$ to a vertex in $\mathcal{L}$ of the same color. All the simplicial maps we consider in this thesis are chromatic. We can now define the concepts of a *subdivision* of a complex, and the *carrier* of a simplex in a subdivision.

Definition 3.13 *Let $\mathcal{K}$ be a complex in R^l. A complex $\sigma(\mathcal{K})$ is said to be a* subdivision *of $\mathcal{K}$ if the following two conditions hold:*

- *Each simplex in $\sigma(\mathcal{K})$ is contained in a simplex in $\mathcal{K}$.*
- *Each simplex of $\mathcal{K}$ equals the union of finitely many simplexes in $\sigma(\mathcal{K})$.*

An example of a complex and its subdivision is given in Figure 3-5.

Definition 3.14 *If S is a simplex of $\sigma(\mathcal{K})$, the* carrier *of S, denoted $carrier(S)$ is the unique smallest $T \in \mathcal{K}$ such that $S \subset |T|$.*

The concept of a carrier of a simplex is illustrated in Figure 3-6. The original complex is shown on the right, and the subdivided complex is shown on the left. A simplex S in the subdivision and the corresponding carrier $carrier(S)$ in the original complex are highlighted in the figure.

A *chromatic subdivision* of $(\mathcal{K}, \chi_{\mathcal{K}})$ is a chromatic complex $\sigma(\mathcal{K}), \chi_{\sigma(\mathcal{K})})$ such that $\sigma(\mathcal{K})$ is a subdivision of $\mathcal{K}$, and for all S in $\sigma(\mathcal{K})$, $\chi_{\sigma(\mathcal{K})}(S) \subseteq \chi_{\mathcal{K}}(carrier(S))$. A simplicial map

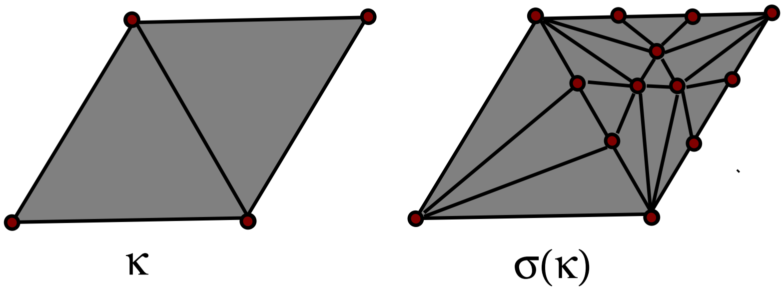

Figure 3-5: Example of a pure, 2-dimensional simplicial complex and a subdivision of it.

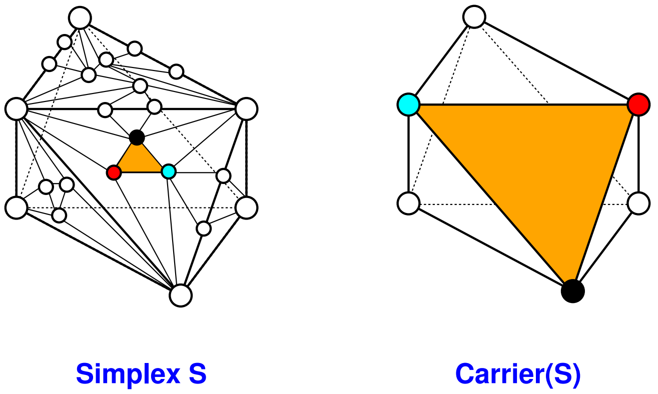

Figure 3-6: The Carrier of a Simplex

$\mu : \sigma_1(\mathcal{K}) \to \sigma_2(\mathcal{K})$ between chromatic subdivisions of $\mathcal{K}$ is *carrier preserving* if for all $S \in \sigma_1(\mathcal{K})$, $carrier(S) = carrier(\mu(S))$. All subdivisions we consider in this thesis will be chromatic, unless explicitly stated otherwise.

3.2 Topological Modeling of Decision Tasks

Earlier in this section, we defined the notion of a decision task in terms of input and output vectors. That definition was intended to help the reader understand what a decision task is, but it lacks the mathematical structure necessary to prove interesting results. We now reformulate this definition in terms of simplicial complexes.

To illustrate our constructions, we will use the well-known Unique-Id task as an informal example. We will give a formal, topological definition of decision tasks later. The *Unique-Id* task was introduced in Section 1, but we restate it here for convenience.

Example 3.15 *The* $n+1$*-process* Unique-Id *task is defined as follows:*

- $I = \{[0, \dots, n]\}$.
- $O = \{[x_0, \dots, x_n] \mid \forall i, j : x_i \in (\mathbb{Z})_{n+1} \cup \{\perp\} \wedge (x_i = x_j) \Rightarrow (x_i = \perp)\}$.
- $\gamma = \{(\vec{I}, \vec{O}) \mid (\vec{I}[i] = \perp \Leftrightarrow \vec{O}[i] = \perp)\}$.

We represent the (unique) $n+1$-dimensional input vector $\vec{I} = [0, \dots, n]$ to the *Unique-Id* task as a simplex S, called an *input simplex*, with dimension $0 \leq \dim(S) \leq n$. The dimension of S equals the number of non-$\perp$ elements in the vector. Each vertex $\vec{v}$ in S is labeled with a process id and an input value: $\langle i, v_i \rangle$, where $\vec{I}[i] = v_i$. We use $ids(S)$ to denote the simplex's set of process ids, and $vals(S)$ to denote the multiset of values. If $\vec{J}$ is a prefix of $\vec{I}$, then the simplex corresponding to $\vec{J}$ is a face of S. The set I of input vectors is thus modeled as a complex $\mathcal{I}$ of input simplexes, called the *input complex*.

Similarly, we represent each $n+1$-dimensional output vector $\vec{O} = [x_1, \dots, x_n]$, where for all i, j, either $x_i = \perp$ or $0 \leq x_i \leq n$ and $(x_i = x_j) \Rightarrow (x_i = \perp)$, as a simplex T, called an *output simplex*, with dimension $0 \leq \dim(T) \leq n$. Each vertex $\vec{v}$ in T is labeled with a process id and an output value: $\langle i, v_i \rangle$, where $\vec{O}[i] = v_i$. We use $ids(T)$ to denote the simplex's set of process ids, and $vals(T)$ to denote the multiset of values. If $\vec{P}$ is a prefix of $\vec{O}$, then the simplex corresponding to $\vec{P}$ is a face of T. The set O of input vectors is thus modeled as a complex $\mathcal{O}$ of output simplexes, called the *output complex*.

The task specification map γ induces a *topological task specification map* Γ for the *Unique-Id* task in the natural way, mapping each input simplex $S \in \mathcal{I}$ to a set $\Gamma(S)$ of output simplexes in $\mathcal{O}$, with the property that for all $T \in \Gamma(S)$, the set $vals(T)$ contains no non-$\perp$ duplicates.

We can now give an alternative, topological representation of the *Unique-Id* decision task by simply specifying it as a tuple $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ consisting of an input complex $\mathcal{I}$, and output complex $\mathcal{O}$, and a topological task specification map, Γ.

Figure 3-7 shows two triangles (2-simplexes) corresponding to two distinct outputs for the 3-process *Unique-Id* task, one given by the output vector $\vec{O}_1 = [0, 1, 2]$, the other given

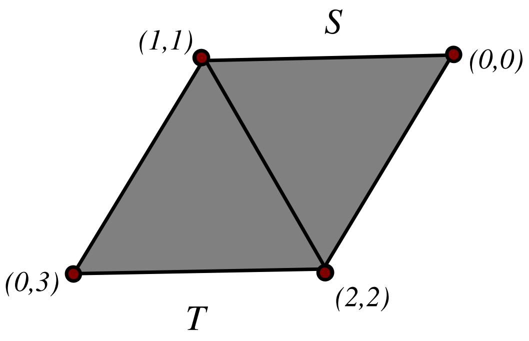

Figure 3-7: Part of output complex for the 3-proc Unique-Id task.

by the output vector $\vec{O}_2 = [3, 1, 2]$. Notice that the vertexes of each simplex are colored by the appropriate process ids.

This topological representation gives an alternative interpretation of the notion of "similar" system states. The processes corresponding to vertexes on the common boundary of the two simplexes cannot distinguish between the two global output sets based on their own output values. Unlike graph-theoretic models (e.g., [6]), simplicial complexes capture in a natural way the notion of the *degree* of similarity between the two global output sets: it is the dimension of the intersection of the two 2-simplexes.

We now give a novel, formal procedure for how to specify a given decision task $\mathcal{D} = \langle I, O, \gamma \rangle$ topologically. We first construct a representation using abstract simplexes and complexes. It then follows from Theorem 3.9 that there exists a representation using geometric simplexes and complexes, for which the vertex scheme is isomorphic to the abstract representation. There are standard ways of constructing such geometric complexes [27], but we choose not to get into the details of these constructions in this thesis.

Definition 3.16 *Let $\vec{I} \in I$ be an input vector. The* input simplex *corresponding to $\vec{I}$, denoted $S(\vec{I})$, is the abstract simplex $(\langle i_0, v_{i_0} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, where for all i_j, where $i_0 \leq i_j \leq i_m$, $i_j \in \{0, \ldots, n\} \wedge \vec{I}[i_j] \neq \perp$, and for all i, $(\vec{I}[i] = \perp) \Rightarrow (i \notin \{i_0, \ldots, i_m\})$.*

Definition 3.17 *Let $\vec{O} \in O$ be an output vector. The* output simplex *corresponding to $\vec{O}$, denoted $T(\vec{O})$, is the abstract simplex $(\langle i_0, v_{i_1} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, where for all i_j, where*

$i_0 \le i_j \le i_m$, $i_j \in \{0, \dots, n\} \wedge \vec{O}[i_j] \neq \perp$, *and for all* i, $(\vec{O}[i] = \perp) \Rightarrow (i \notin \{i_0, \dots, i_m\})$.

Having defined input and output simplexes, we can define input and output complexes.

Definition 3.18 *The* input complex *corresponding to* I*, denoted* $\mathcal{I}$*, is the collection of input simplices* $S(\vec{I})$ *corresponding to the input vectors of* I*.*

Definition 3.19 *The* output complex *corresponding to* O*, denoted* $\mathcal{O}$*, is the collection of output simplices* $T(\vec{O})$ *corresponding to the output vectors of* O*.*

We have to prove that these definitions make sense topologically, that is, that the input and output complexes thus defined are in fact abstract complexes. We will do the proof for input complexes in the lemma below. The proof relies on the requirement that the sets of input vectors we consider are prefix-closed.

Lemma 3.20 *Given a set* I *of input vectors. The corresponding input complex* $\mathcal{I}$*, as defined in Definition 3.18, is an abstract, chromatic complex.*

Proof. Clearly, $\mathcal{I}$ as defined above is a collection of abstract simplexes. We must show that this collection satisfies the requirements of Definition 3.6, that is, we must establish that $\mathcal{I}$ is closed under containment.

Let $S(\vec{I}) = (\vec{s}_0, \dots, \vec{s}_m)$ be an abstract simplex of $\mathcal{I}$, and let $T = (\vec{t}_0, \dots, \vec{t}_{m'})$, where $m' \le m$, be a face of $S(\vec{I})$. It follows that $skel^0(T) \subseteq skel^0(S(\vec{I}))$. Consider the vector $\vec{J}$, the i-th entry of which is $\perp$ iff $i \notin ids(T)$, and otherwise $val(\vec{t}_j)$, where $i = id(\vec{t}_j)$. We claim that this vector is a prefix of $\vec{I}$. Suppose $\vec{I}[i] = \perp$. Then $i \notin ids(S(\vec{I}))$, and hence $i \notin ids(T)$. It follows that $\vec{J}[i] = \perp$. Now suppose that $\vec{J}[i] \neq \perp$. Then there is a vertex $\vec{t}_j$ in T with $id(\vec{t}_j) = i$ and $val(\vec{t}_j) = \vec{J}[i]$. Since $skel^0(T) \subseteq skel^0(S(\vec{I}))$, this vertex is also a vertex of $S(\vec{I})$, and so it follows that $\vec{I}[i] = val(\vec{t}_j) = \vec{J}[i]$. It follows that $\vec{J}$ is a prefix of $\vec{I}$. Since we only consider prefix-closed sets of input vectors, $\vec{J}$ must be in I. The simplex corresponding to $\vec{J}$, denoted $S(\vec{J})$ is equal to T, and hence T is in $\mathcal{I}$.

It follows that $\mathcal{I}$ is an abstract complex. Moreover, let $S = (\vec{s}_0, \dots, \vec{s}_n)$ be any n-simplex. Then the map $\chi : \mathcal{I} \to \mathcal{S}$ defined by $\chi(\vec{v}) = \vec{s}_i$, where $i = id(\vec{v})$ is a chromatic and simplicial map from $\mathcal{I}$ to $\mathcal{S}$, and hence $\mathcal{I}$ is a chromatic, abstract complex, as required. □

Lemma 3.21 *Given a set O of output vectors. The corresponding output complex $\mathcal{O}$, as defined in Definition 3.18, is an abstract, chromatic complex.*

The proof is identical to the proof of Lemma 3.20, and will be omitted here. Given a pair of (abstract) input and output complexes, we may apply Theorem 3.9 to construct a corresponding pair of geometric chromatic input and output complexes by embedding the abstract complexes in $\mathbb{R}^{2n+1}$. As discussed in Section 3.1, the abstract and geometric representations of a complex are equivalent up to linear isomorphism, and thus we will work with both interchangeably in the remainder of this thesis.

We now construct a topological equivalent of the task specification map $\gamma \subseteq I \times O$.

Definition 3.22 *The* topological task specification map *corresponding to γ, denoted $\Gamma \subseteq \mathcal{I} \times \mathcal{O}$, is defined as follows.*

$$(S(\vec{I}), T(\vec{O})) \in \Gamma \Longleftrightarrow (\vec{I}, \vec{O}) \in \gamma$$

As a convenient notation, for all $S(\vec{I}) \in \mathcal{I}$, we denote the set of simplexes $T(\vec{O})$ in $\mathcal{O}$ such that $(S(\vec{I}), T(\vec{O})) \in \Gamma$ by $\Gamma(S(\vec{I}))$. Usually, we simply refer to a topological task specification map as a "task specification map". We now prove that task specifications are id-preserving; if a process i has an input value, it must also have an output value, and vice versa.

Lemma 3.23 *For all $S(\vec{I}) \in \mathcal{I}$, and all $T(\vec{O}) \in \Gamma(S(\vec{I}))$, $ids(T) = ids(S)$.*

Proof. Let $S(\vec{I})$ be any simplex in $\mathcal{I}$, and let $T(\vec{O}) \in \Gamma(S(\vec{I}))$. Then $\vec{O} \in \gamma(\vec{I})$ by Definition 3.22. Suppose $i \notin ids(S(\vec{I}))$. Then $\vec{I}[i] = \perp$ by Definition 3.16, and hence by Definition 2.6, $\vec{O}[i] = \perp$. It follows from Definition 3.17 that $i \notin ids(T(\vec{O}))$. Now suppose $i \notin ids(T(\vec{O}))$. Then $\vec{O}[i] = \perp$ by Definition 3.17, and hence by Definition 2.6, $\vec{I}[i] = \perp$. It follows from Definition 3.16 that $i \notin ids(S(\vec{I}))$. $\square$

An illustration of a pair of input and output complexes, together with a task specification map relating them, is given in Figure 3-8.

Definition 3.24 *Given a decision task $\mathcal{D} = \langle I, O, \gamma \rangle$, the corresponding* topological representation *of the task, denoted $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, consists of an input complex $\mathcal{I}$ corresponding to I, and output complex $\mathcal{O}$ corresponding to O, and a task specification map Γ corresponding to γ.*

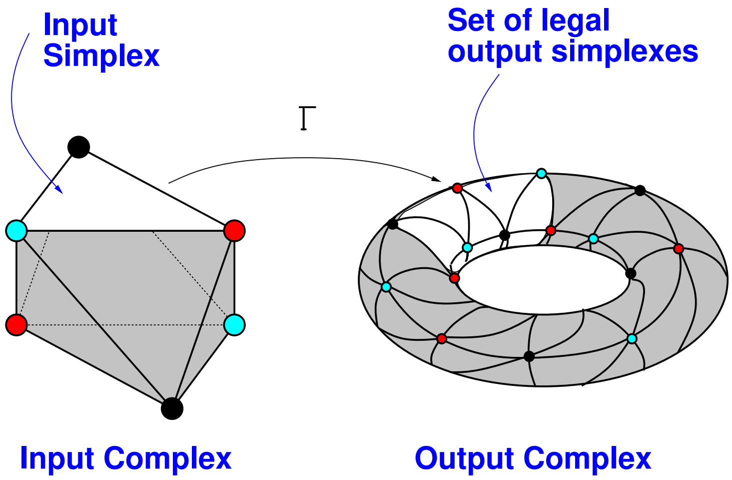

Figure 3-8: A Decision Task

In the remainder of this paper, we will specify decision tasks using both Definition 2.7 and Definition 3.24 interchangeably. A set of inputs or outputs may thus be specified as either a vector or as a simplex the vertexes of which are labeled with process ids and values.

3.3 Topological Modeling of NIIS Protocols

We model protocols in the NIIS model in much the same way that we model decision tasks. As discussed in Chapter 2, the sets of inputs and outputs for any execution α of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model can be modeled using $n+1$ process input and output vectors. We denote the sets of input vectors and output vectors of a protocol by I and O, respectively. We are only interested in protocols that solve decision tasks, so we may assume that the set I of possible input vectors to a protocol is prefix-closed. The following lemma states that for any protocol in the NIIS model, the set O of possible output vectors from all executions of the protocol must necessarily be prefix-closed. Recall from Section 2.5 that we are only considering the set of fair (and hence finite) executions of a protocol here.

Lemma 3.25 *Let O be the set of possible output vectors of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, with corresponding set of input vectors I. Then O is prefix-closed.*

Proof. Let $\vec{O}$ be an output vector produced by the execution α_O, and let $\vec{P}$ be a prefix of $\vec{O}$. We construct an execution α_P as follows: For each i such that $\vec{O}[i] = v_i \neq \perp = \vec{P}[i]$, replace the action *decide(S)*$_i$ (S is the snapshot returned by that action) in α_O with a *fail*$_i$ action, meaning that process i fail-stopped before deciding. Clearly, the execution thus obtained is a possible execution of $\mathcal{P}_{(n,\tau,\delta)}$, and its output vector is $\vec{P}$. Hence $\vec{P}$ is in O, and O is prefix-closed. $\square$

Given that both the set of input vectors I and the set of output vectors O associated with a protocol $\mathcal{P}_{(n,\tau,\delta)}$ are prefix-closed sets of vectors, we can construct corresponding input and output complexes, denoted $\mathcal{I}$ and $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$, respectively. These complexes are constructed in the same way as the complexes corresponding to input and output sets of vectors for decision tasks, and the proofs that they are indeed chromatic complexes are also identical. The output complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$ is called a *protocol complex.*

Let $\mathcal{J}$ be a subcomplex of the input complex $\mathcal{I}$. The set of possible outputs when the protocol is given inputs corresponding to simplexes in $\mathcal{J}$ is denoted $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$.

Lemma 3.26 *Let $\mathcal{J}$ be a subcomplex of $\mathcal{I}$. Then $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$.*

Proof. It suffices to show that $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a complex, since $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is clearly a subset of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. We simply look at the set of vectors J corresponding to the subcomplex $\mathcal{J}$ in isloation, as the set of input vectors to the protocol $\mathcal{P}_{(n,\tau,\delta)}$. This set is prefix-closed since $\mathcal{J}$ is a complex, and hence closed under containment. Hence the set P of output vectors given input vecors in J is prefix-closed by Lemma 3.25. It follows from Lemma 3.21 that $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$, the complex corresponding to $\mathcal{P}_{(n,\tau,\delta)}$, is a complex, and hence a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. $\square$

In the remainder of this thesis, we will specify protocols in NIIS using both its formal specification from Section 2.5 as well as protocol complexes as described in this section interchangeably. A set of inputs or outputs may thus be specified as either a vector or as a simplex the vertexes of which are labeled with process ids and values.

3.4 Subdivisions

The standard chromatic subdivision was introduced by Herlihy and Shavit as part of their work on asynchronous computability [21, 22, 26]. It is essentially a chromatic generalization of the standard barycentric subdivision from classical algebraic topology [27, 31]. In this section, we will present a complete, formal definition of the standard chromatic subdivision, together with a proof that this definition does indeed correspond to a chromatic subdivision of a given complex. Our definition is somewhat different from that of Herlihy and Shavit [21, 22, 26], as it is based on an explicit, inductive, geometric construction, which we later formally prove to be a chromatic subdivision. We also introduce the concept of a *non-uniform chromatic subdivision*, a generalization of the standard chromatic subdivision, in which the different simplexes of a complex are not necessarily subdivided the same number of times. Informally, a non-uniform chromatic subdivision of level 1 of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^1(\mathcal{K})$, is constructed by choosing, for each n-simplex in $\mathcal{K}$, a *single* face of the simplex (a face can be of any dimension and could also be the whole simplex) to which we apply the standard chromatic subdivision. We then induce the subdivision onto the rest of the simplex. The subdivisions of any two individual simplexes must be such that they agree on their shared face. This can be seen in Figure 3-9. Its right hand side shows a valid non-uniform chromatic subdivision of a complex where, for example, the simplex (b, c, d)'s subdivision is the result of subdividing the 1-face (c, d) once and then inducing this subdivision onto the rest of the simplex. The left hand side structure is not a legal subdivision, since the subdivision of the simplex (b, c, d) does not agree with that of the simplex (a, b, d) on the shared face (b, d). This structure is not even a simplicial complex, since it contains an object that is not a simplex (the cross-hatched region in Figure 3-9. A k-th level non-uniform chromatic subdivision of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^k(\mathcal{K})$, is generated by repeating this process k times, where only simplexes in faces that were subdivided in round $k-1$ can be subdivided in phase k. The complex on the right hand side of Figure 3-9 is an example of a non-uniform chromatic subdivision of level 2, since the face (a, d) is subdivided twice.

Later, we will show that these structures correspond in a natural way to the set of protocol complexes in the NIIS model of computation. In fact, it turns out that each non-uniform standard chromatic subdivision is equal to some NIIS protocol complex (up to

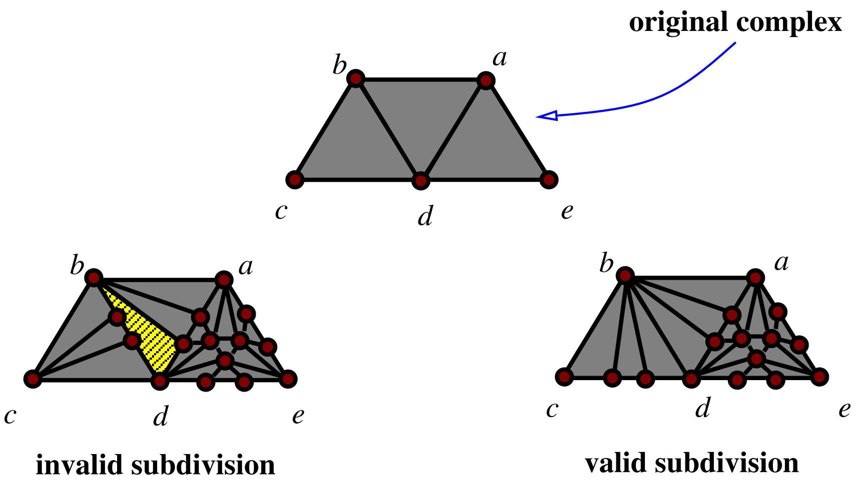

Figure 3-9: Valid and Invalid Non-uniform Subdivisions

isomorphism).

3.4.1 The Standard Chromatic Subdivision

In this section we provide our definition of the *standard chromatic subdivision*, and prove that this definition does indeed specify a chromatic subdivision of a given complex. While Herlihy and Shavit gave a high level combinatorial definition of the standard chromatic subdivision [21, 22, 26], our definition is based on an explicit, geometric construction.

Let $\mathcal{K}$ be a pure, n-dimensional, chromatic geometric complex, where the colors are the numbers in $\mathbb{Z}_{n+1}$. Label each vertex $\vec{v}$ in $\mathcal{K}$ with $\langle i, v_i \rangle$, where i is the color of $\vec{v}$, and v_i is a value in some set D_I chosen such that no two vertexes in $\mathcal{K}$ have the same label. In order to define the standard chromatic subdivision of $\mathcal{K}$, we inductively define a sequence of subdivisions $\mathcal{L}_p$ of the skeletons of $\mathcal{K}$, where $0 \leq p \leq n$ as follows. Let $\mathcal{L}_0 = skel^0(\mathcal{K})$. Now suppose that $\mathcal{L}_{p-1}$ is a chromatic subdivision of the $p-1$-skeleton of $\mathcal{K}$. Each vertex $\vec{v}$ in $\mathcal{L}_{p-1}$ is labeled $\langle i, S_i \rangle$, where S_i is the vertex scheme of some simplex in $skel^{p-1}(\mathcal{K})$. The labels $\langle i, S_i \rangle$ are such that any $T = (\vec{t_0}, \ldots, \vec{t_r})$, where $r \leq p-1$, is a simplex in $\mathcal{L}_{p-1}$ iff $ids(T) \subseteq ids(carrier(T)$ and the following conditions hold for all $1 \leq i, j \leq r$:

- $id(t_i) \neq id(t_j)$.
- $id(\vec{t_i}) \in ids(val(\vec{t_i}))$
- $val(\vec{t_i})$ is a face of $val(\vec{t_j})$ or vice versa

- $id(\vec{t_j}) \in ids(val(\vec{t_i})) \Rightarrow val(\vec{t_j})$ is a face of $val(\vec{t_i})$

Let $S = (\vec{s_0}, \ldots, \vec{s_p})$ be a p-simplex in $\mathcal{K}$. The set $Bd(S)$ is the polytope of a subcomplex of the $p-1$-skeleton of $\mathcal{K}$, and hence of a subcomplex of $\mathcal{L}_{p-1}$, which we denote $\mathcal{L}_{Bd(S)}$. Let $\vec{b}$ be the barycenter of S, and let δ be some positive real number such that $0 < \delta < 1$. For each $1 \leq i \leq p$, define $\vec{m_i}$ to be the point $(1+\delta)\vec{b} - \delta\vec{s_i}$. These points are called the midpoints of S. Label $\vec{m_i}$ with $\langle i, S\rangle$, S here being the vertex scheme of the geometric simplex S. Let M_S be the set of midpoints of S. We define $\mathcal{L}_S$ to be the union of $\mathcal{L}_{Bd(S)}$ and all the faces of all chromatic p-simplexes $T = (\vec{t_0}, \ldots, \vec{t_p})$, such that for all $1 \leq i, j \leq p$: $\vec{t_i} \in skel^0(\mathcal{L}_{Bd(S)}) \cup M_S$, and the following conditions hold:

- $id(t_i) \neq id(t_j)$.
- $id(\vec{t_i}) \in ids(val(\vec{t_i}))$
- $val(\vec{t_i})$ is a face of $val(\vec{t_j})$ or vice versa
- $id(\vec{t_j}) \in ids(val(\vec{t_i})) \Rightarrow val(\vec{t_j})$ is a face of $val(\vec{t_i})$

We now define $\mathcal{L}_p$ to be the complex consisting of the union of the complexes $\mathcal{L}_S$, as S ranges over all the p-simplexes of $\mathcal{K}$. We now prove that this structure makes sense mathematically, that is, that it is in fact a subdivision of the p-skeleton of $\mathcal{K}$.

Lemma 3.27 *For all $0 \leq p \leq n$, $\mathcal{L}_p$ is a chromatic subdivision of $skel^p(\mathcal{K})$.*

Proof. We argue by induction. The case $p = 0$ is trivial. So suppose $p > 0$, and suppose the claim holds for $\mathcal{L}_0, \ldots, \mathcal{L}_{p-1}$. We will first prove that $\mathcal{L}_p$ is a chromatic simplicial complex. To that end, we prove the following auxiliary lemma.

Lemma 3.28 *For all p-simplexes S in $\mathcal{K}$, $\mathcal{L}_S$ is a chromatic complex.*

Proof. We must show that $\mathcal{L}_S$ is closed under containment and intersection. Let U be a simplex in $\mathcal{L}_S$, and let V be a face of U, where $0 \leq \dim(V) \leq \dim(U) \leq p$. If U is in $\mathcal{L}_{Bd(S)}$, then so is V, since $\mathcal{L}_{Bd(S)}$ is a complex (since $\mathcal{L}_p$ is a subdivision and hence a complex by assumption). Hence V is in $\mathcal{L}_S$. Suppose U is not contained in $\mathcal{L}_{Bd(S)}$. Then U must be the face of a p-simplex T as described above. By definition of $\mathcal{L}_S$, all the faces of T, and hence all faces of U, must be in $\mathcal{L}_S$. It follows that $\mathcal{L}_S$ is closed under containment.

Let U, V be simplexes in $\mathcal{L}_S$, and suppose their intersection, denoted by W, is nonempty. If U, V are both in $\mathcal{L}_{Bd(S)}$, it follows immediately that V^r is in $\mathcal{L}_{Bd(S)}$ and hence in L_S. Similarly, if U is in $\mathcal{L}_{Bd(S)}$ but V is not, then $W = U \cap V = U \cap (V \cap |\mathcal{L}_{Bd(S)}|)$. Note that $V \cap |\mathcal{L}_{Bd(S)}|$ is a simplex in $\mathcal{L}_{Bd(S)}$, since all the criteria given above are satisfied. Hence it follows that W is in $\mathcal{L}_{Bd(S)}$, and hence in $\mathcal{L}_S$. If neither U nor V is in $\mathcal{L}_{Bd(S)}$, then since all faces of U and V are in $\mathcal{L}_S$, then so is W. It follows that $\mathcal{L}_S$ is closed under intersection, and hence is a complex. That $\mathcal{L}_S$ is chromatic follows from the fact that we only include chromatic simplexes in $\mathcal{L}_S$ in our construction (note that $\mathcal{L}_{p-1}$ and hence $\mathcal{L}_{Bd(S)}$ are chromatic by assumption). $\square$

Notice that for all distinct p-simplexes S, T we have that $|\mathcal{L}_S| \cap |\mathcal{L}_T| = S \cap T$, which is a simplex in $skel^{p-1}(\mathcal{K})$, and hence is the polytope of a subcomplex of $\mathcal{L}_{p-1}$, and hence of both $\mathcal{L}_S$ and $\mathcal{L}_T$. It follows that $\mathcal{L}_p$ is a simplicial complex [27]. It remains to show that $\mathcal{L}_p$ is a chromatic subdivision. To this end, we must first show that every simplex in $\mathcal{L}_p$ is contained in some simplex in $skel^p(\mathcal{K})$, and that every simplex in $skel^p(\mathcal{K})$ is the union of finitely many simplexes in $\mathcal{L}_p$. Now, it is clear from our construction that any simplex T_q in $\mathcal{L}_p$ is contained in some simplex S in $skel^p(\mathcal{K})$. Also, since for all simplexes S in $skel^p(\mathcal{K})$, the set of midpoints is finite, and $\mathcal{L}_{p-1}$ is a subdivision of $skel^{p-1}(\mathcal{K})$ by assumption, it follows that S is the union of finitely many simplexes in $\mathcal{L}_p$. Hence $\mathcal{L}_p$ is a subdivision. This subdivision is chromatic, since $\mathcal{L}_{p-1}$ is chromatic by assumption, since the colors used to color the midpoints of any simplex S are exactly the colors used to color S, and since any simplex in $\mathcal{L}_p$ including midpoints must satisfy the requirement that no two vertexes have the same color (id). $\square$

We are now ready to give our definition of the standard chromatic subdivision of a complex $\mathcal{K}$.

Definition 3.29 *The* standard chromatic subdivision *of $\mathcal{K}$, denoted $\mathcal{X}(\mathcal{K})$, is the complex $\mathcal{L}_n$.*

An example of a complex and its standard chromatic subdivision is given in Figure 3-10.

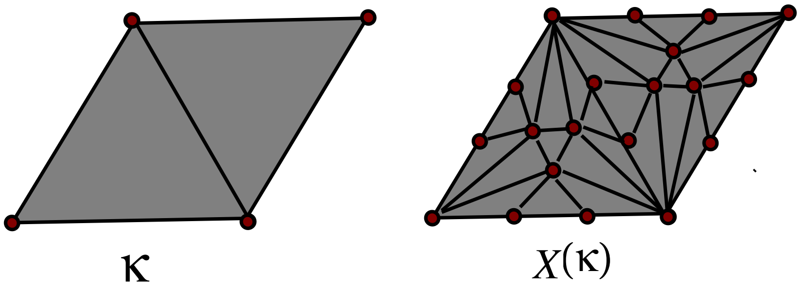

Figure 3-10: Example of a 2-dimensional complex and its standard chromatic subdivision.

Applying the standard chromatic subdivision k times, where $k > 1$, yields a subdivision $\mathcal{X}^k(\mathcal{K}) = \mathcal{X}^{k-1}(\mathcal{X}(\mathcal{K}))$, which we call the kth iterated standard chromatic subdivision [21, 22, 26]. Since the standard chromatic subdivision of a complex is again a complex, and a chromatic subdivision of a chromatic subdivision of $\mathcal{K}$ is itself a chromatic subdivision of $\mathcal{K}$, $\mathcal{X}^k(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$. The number k is called the *level* of the subdivision. Moreover, for all k, $\mathcal{X}^k(\mathcal{K}) = \mathcal{X}^{k-1}(\mathcal{X}(\mathcal{K})) = \mathcal{X}(\mathcal{X}^{k-1}(\mathcal{K}))$.

We will mostly use the vertex scheme representation of the standard chromatic subdivision, as it has a particularly compact representation, as stated in the lemma below. We note that this formulation of the standard chromatic subdivision is equivalent to the definition of Herlihy and Shavit [21, 22, 26].

Lemma 3.30 *Let $\mathcal{K}$ be a pure, chromatic complex of dimension n. The vertex scheme of $\mathcal{X}(\mathcal{K})$ is the closure under containment of the set of all n-simplexes of the form $S = (\langle 0, S_0\rangle, \ldots, \langle n, S_n\rangle)$, where for all i, S_i is the vertex scheme of some face of a simplex S in $\mathcal{K}$, and the following conditions hold for all i, j:*

- *$i \in ids(S_i)$.*
- *S_i is a face of S_j or vice versa.*
- *If $j \in ids(S_i)$, then S_j is a face of S_i.*

Proof. We argue by induction on k, where $0 \le k \le n$. It is immediate that the simplexes of $\mathcal{X}(\mathcal{K})$ lying in the subdivision $\mathcal{L}_0$ of $skel^0(\mathcal{K})$ are of this form (each such simplex is a vertex of $\mathcal{K}$ labeled with a process id and a value), and the requirements above are all satisfied trivially.

Now suppose the claim holds for $0, \ldots, k-1$. Consider a simplex T lying in the subdivision $\mathcal{L}_k$ of $skel^k(\mathcal{K})$ and not in $\mathcal{L}_{k-1}$. Then $T = U \star V$, where U is a simplex in $\mathcal{L}_{k-1}$, and V is a simplex each vertex of which is one of the midpoints in M_S, where $S = carrier(T)$. By assumption, V is non-trivial, meaning that there is at least one vertex in V. However, U may be trivial. For each vertex $\vec{v}$ in V, $val(\vec{v}) = S$, the vertex scheme of S. Hence for i, j in $ids(V)$, since $ids(V) \subseteq ids(S)$, all the conditions listed above are clearly satisfied. For i, j in $ids(U)$, the conditions are satisfied by induction. Now suppose i is in $ids(U)$, while j is in $ids(V)$. Notice that $S_i = carrier(U)$, and $S_j = carrier(V) = S$.

The first condition follows by induction (for i) and since $ids(V) \subseteq ids(S) = vals(V)$ (for j). Since $carrier(U)$ is a face of S, it follows that S_i is a proper face of S, and hence of S_j, which equals the vertex scheme of S, and so the second condition is satisfied. It is clear that i is in $ids(S_j)$, since S_j equals the vertex scheme of S, and i is in $ids(carrier(U))$, which is a subset of $ids(S)$. That S_i is a face of S_j has already been established. It follows that, since $\mathcal{X}(\mathcal{K})$ is chromatic, $ids(U) \cap ids(V) = emptyset$, $ids(S_i) \subseteq ids(carrier(U))$, and j is not in $ids(carrier(U))$, j cannot be in $ids(S_i)$. It follows that the third condition is satisfied. $\square$

In the remainder of this thesis, we will usually work with this description of the standard chromatic subdivision, and refer to it as $\mathcal{X}(\mathcal{K})$. Whenever the distinction between the geometric and abstract representations of $\mathcal{X}(\mathcal{K})$ is significant, it will be mentioned explicitly.

3.4.2 The Non-Uniform Chromatic Subdivision

In this section, we define the *non-uniform chromatic subdivision*, and prove that this definition does indeed specify a chromatic subdivision of a given complex.

Definition 3.31 *Let $\mathcal{K}$ be a pure n-dimensional chromatic complex, where the colors are the numbers in $\mathcal{Z}_{n+1}$. We will give a recursive definition of a* non-uniform chromatic subdivision *of $\mathcal{K}$. In general, we denote a k-level non-uniform chromatic subdivision of a complex $\mathcal{K}$ by $\widetilde{\mathcal{X}}^k(\mathcal{K})$, for $k \geq 0$.*

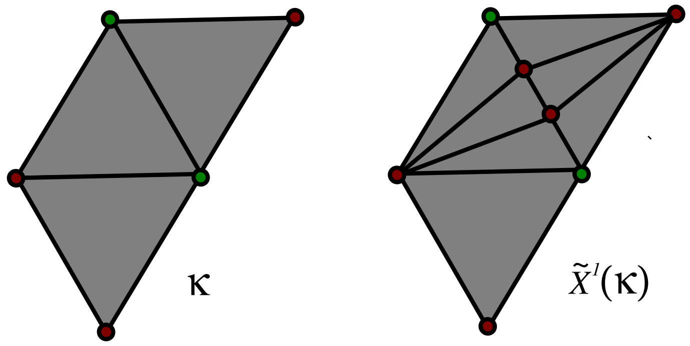

Figure 3-11: Example of level 1 non-uniform chromatic subdivision of a 2-complex.

If $\dim(\mathcal{K}) = 0$, *then for all* $k \geq 0$, $\tilde{\mathcal{X}}^k(\mathcal{K})$ *is* $\mathcal{K}$ *itself. Now suppose* $\dim(\mathcal{K}) > 0$. *Then* $\tilde{\mathcal{X}}^0(\mathcal{K})$ *is* $\mathcal{K}$ *itself. For* $k > 0$, $\tilde{\mathcal{X}}^k(\mathcal{K})$ *is given by the following procedure: Partition the vertexes of* $\mathcal{K}$ *into two disjoint sets,* A *and* B, *where* $\mathcal{A}$ *is nonempty. From these sets we construct two complexes, denoted by* $\mathcal{A}$ *and* $\mathcal{B}$, *respectively, as follows: A simplex* S *in* $\mathcal{K}$ *is in* $\mathcal{A}$ *if the vertexes spanning* S *are all in* A, *and it is in* $\mathcal{B}$ *if its spanning vertexes are all in* $\mathcal{B}$. *The subdivision* $\tilde{\mathcal{X}}^k(\mathcal{K})$ *is the complex consisting of all simplexes in* $\mathcal{B}$, *all simplexes in* $\tilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, *and all simplexes of the form* $S \star T$, *where* S *is a simplex in* $\tilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T *is a simplex in* $\mathcal{B}$, *and* $carrier(S) \star T$ *is a simplex in* $\mathcal{K}$.

Informally speaking, a non-uniform chromatic subdivision of level k is one in which there is some simplex in $\mathcal{K}$ which is subdivided k times, but no simplex that is subdivided more than k times. Note that the k level standard chromatic subdivision is a special case of the k level non-uniform chromatic subdivision. Hence for all $k \geq 0$, there exists some non-uniform chromatic subdivision of level k. Our definition of non-uniform chromatic subdivisions is designed to easily model protocol complexes of the NIIS model. At any level of the recursion, the vertexes in $\mathcal{A}$ can be thought of as corresponding to processes that continue computing given their current local state, while the vertexes in $\mathcal{B}$ correspond to processes that decide. An example of a level 1 non-uniform chromatic subdivision of a 2-complex $\mathcal{K}$ is given in Figure 3-11, and an example of a level 2 non-uniform chromatic subdivision of a slightly bigger 2-complex $\mathcal{L}$ is given in Figure 3-12. Note that in Figure 3-12 the complex $\mathcal{A}$ for the second level of recursion is isomorphic to the complex $\mathcal{A}$ for the first level of recursion in Figure 3-11.

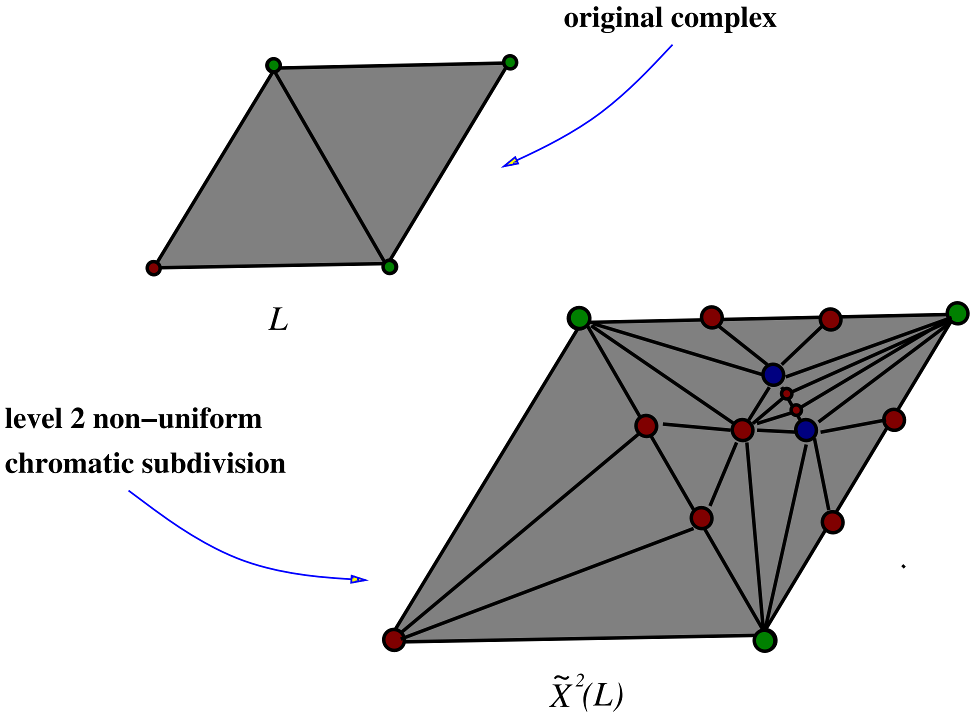

Figure 3-12: Example of level 2 non-uniform chromatic subdivision of a 2-complex.

An example of a chromatic subdivision that does *not* satisfy Definition 3.31 is given in Figure 3-13. It is not a non-uniform chromatic subdivision because the vertex b is part of the $\mathcal{B}$ complex at the first level of recursion (that is, it is not part of the subcomplex that is subdivided further), while in the next level of recursion, the edge between d (which is in the $\mathcal{A}$ complex at the first level of recursion, and hence is to be subdivided further) and b is subdivided, meaning that b is in the $\mathcal{A}$ complex at the second level of recursion, which is clearly impossible, since the carrier of any vertex in the $\mathcal{A}$ complex at the second level must be a simplex in the $\mathcal{A}$ complex at the first level of recursion. Informally, this simply means that, if a vertex is not to be part of the complex to be further subdivided at the first level, it cannot be part of the complex to be further subdivided at the second level.

Lemma 3.32 *Any non-uniform chromatic subdivision $\tilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$.*

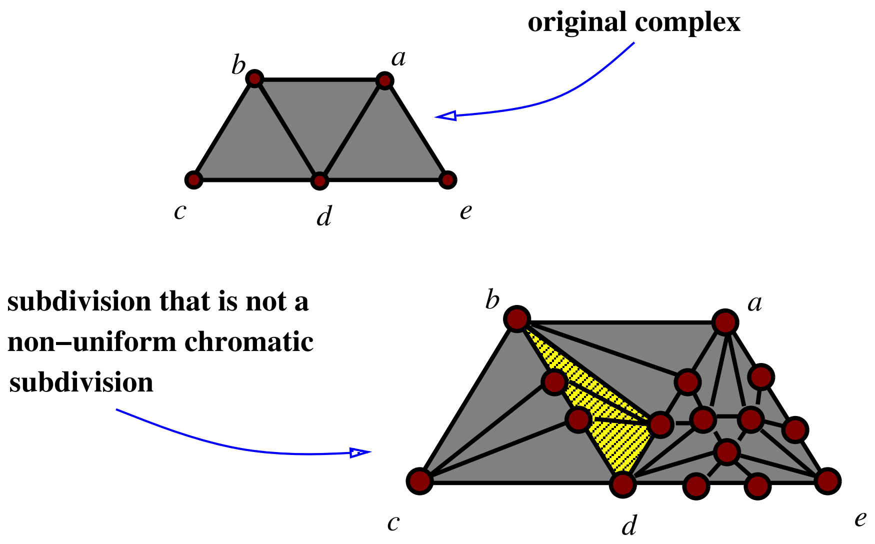

Figure 3-13: Example of a subdivision that is *not* a non-uniform chromatic subdivision.

Proof. We first note that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is well-defined, since each recursive step lowers the level of subdivision by 1, and $\widetilde{\mathcal{X}}^0(\mathcal{K})$ is defined for all $\mathcal{K}$. We will prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision by induction on k.

The case where $k = 0$ is trivial, since $\widetilde{\mathcal{X}}^0(\mathcal{K}) = \mathcal{K}$. Now suppose that $k > 0$, and that for $0 \le l \le k-1$, and any complex $\mathcal{K}$, $\widetilde{\mathcal{X}}^l(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$. If $B = \emptyset$, the result follows by induction and by Lemma 3.27. So suppose that B is nonempty.

We first show that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment. Let U be a simplex in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let V be a face of U. If U is in $\mathcal{B}$, then so is V, since $\mathcal{B}$ is a complex. Hence V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. Similarly, if U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is V, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is a complex by our induction hypothesis. Now suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$. Then $S \cap V$ is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and $T \cap V$ is in $\mathcal{B}$. It follows that $V = (S \cap V) \star (T \cap V)$, where $carrier(S \cap V) \star (T \cap V)$ is a simplex in $\mathcal{K}$. By Definition 3.31, V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. It follows that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment.

Let U, V be simplexes in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let W be their intersection. If both U, V are in $\mathcal{B}$, then so is W, since $\mathcal{B}$ is closed under intersection. Similarly, if both U and V are in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is W, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is closed under intersection. If U is in $\mathcal{B}$ and V is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, or vice versa, then $U \cap V = \emptyset$, and so containment under intersection holds vacuously. We now consider the case where either U or V is not contained in either complex, that is, suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$, and

$V = X \star Y$ for some X in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, Y in $\mathcal{B}$. Now, $U \cap V = (S \star T) \cap (X \star Y)$, and $(S \star T) \cap (X \star Y) = (S \cap X) \star (T \cap Y)$ [27]. If $S \cap X = \emptyset$ or $T \cap Y = \emptyset$, then one of the cases discussed above applies, and V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. So suppose now that S, T, X, Y, and the intersections $S \cap X$ and $Y \cap T$ are all nonempty. Since $carrier(S \cap X)$ is a face of both $carrier(S)$ and $carrier(X)$, it follows that $carrier(S \cap X) \star T$ and $carrier(S \cap X) \star Y$ are simplexes in $\mathcal{K}$, and so is their intersection $carrier(S \cap X) \star (T \cap Y)$, since $\mathcal{K}$ is a complex. It follows that V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. This concludes the proof that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a complex. That it is a chromatic complex follows directly from Lemma 3.27.

We now prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision. Given a simplex U in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. If U is in $\mathcal{B}$ then U is clearly contained in a simplex in $\mathcal{K}$, namely itself, and that the colors of U are contained in the set of colors of its carrier. If U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, it follows by induction and Lemma 3.27 that U is contained in some some simplex $carrier(U)$ in $\mathcal{A}$, and hence in $\mathcal{K}$, and that the colors of U are a subset of the colors of its carrier. Now suppose $U = S \star T$, where $carrier(S) \star T$ is in $\mathcal{K}$. Then U is contained in $carrier(S) \star T$, and the colors of U are a subset of the colors of $carrier(S) \star T$. Now consider any simplex U in $\mathcal{K}$. We can decompose it into two disjoint faces S and T, such that $S \in \mathcal{A}$ and $T \in \mathcal{B}$. The simplex S is subdivided according to $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, which by induction and Lemma 3.27 consists of finitely many simplexes. The simplex T is not subdivided at all. It follows that the subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ subdivides U into finitely many simplexes (those in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{S})) \star T$). This completes the proof that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision. □

A non-uniform subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ of a complex induces a non-uniform chromatic subdivision of any subcomplex $\mathcal{L}$ of $\mathcal{K}$. The level of the induced subdivision of $\mathcal{L}$ may vary from subcomplex to subcomplex.

Definition 3.33 *Let $\mathcal{K}$ be a chromatic complex, $\mathcal{L}$ a subcomplex or simplex of $\mathcal{K}$, and let $\widetilde{\mathcal{X}}^k(\mathcal{K})$ be a non-uniform iterated chromatic subdivision of $\mathcal{K}$. We denote its restriction to simplexes in $\mathcal{L}$ for $\widetilde{\mathcal{X}}^k(\mathcal{L})$. The* level *of $\widetilde{\mathcal{X}}^k(\mathcal{K})$ on $\mathcal{L}$ is the maximal level of subdivision of $\widetilde{\mathcal{X}}^k(\mathcal{L})$, which we denote by $k_{\mathcal{L}}$.*

It is clear that for any subcomplex or simplex $\mathcal{L}$ of $\mathcal{K}$, it must be the case that the level of $\widetilde{\mathcal{X}}^k(\mathcal{K})$ on $\mathcal{L}$ is less than or equal to k.

Chapter 4

The Asynchronous Complexity Theorem

The strength and usefulness of the NIIS model of computation comes from the fact that each of its associated protocol complexes has a nice, recursive structure. In fact, it turns out that any protocol complex of NIIS is equal to some non-uniform iterated chromatic subdivision of the input complex, and vice versa. This is the essence of our main theorem, which we state and prove in this chapter.

The level of subdivision necessary for the existence of a simplicial map from the input to the output complex of a decision task that agrees with the task specification can be interpreted as a topological measure of the task's time complexity. The following definition introduces the concept of *mappability*, which is a useful construct for reasoning about this topological measure.

Definition 4.1 *Given a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ and a non-negative integer k, we say that $\widetilde{\mathcal{X}}^k(\mathcal{I})$ is a* mappable *subdivision of the input complex, and k is a* mappable *level of subdivision if there exists some chromatic simplicial map μ from $\widetilde{\mathcal{X}}^k(\mathcal{I})$ to $\mathcal{O}$ such that for all T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T) \in \Gamma(T)$.*

This definition extends naturally to individual simplexes as the map induces different levels of subdivision on the individual simplexes in accordance with the idea that, in order to solve a decision task, some processes may have to do more computational work than others, and some inputs may require more computation than others. We can now state our main theorem:

Theorem 4.2 (Time Complexity) *A decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ has a wait-free solution protocol in the NIIS model with worst case time complexity k_S on inputs in S, where S is a simplex in $\mathcal{I}$, if and only if there is a mappable non-uniform iterated chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ with level k_S on S.*

Keeping in style with Herlihy and Shavit [21, 22, 26], the theorem simply states that solvability of a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ in the NIIS model is equivalent to the existence of a chromatic simplicial map μ from some non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ to $\mathcal{O}$ that agrees with the task specification Γ, that is, for all T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T^m) \in \Gamma(T)$. The level k_S is a lower bound on the worst case time complexity of solving this task with inputs in S in the NIIS model.

The theorem also immediately provides a matching upper bound given the subdivision and simplicial mapping. Simply run the normal form protocol of Figure 2-7. Since each process can locally store the subdivision and mapping, the termination predicate map τ just needs to test if the *local_state* variable is equal to some node v in the subdivision and if so return $\mu(v)$.

In the remainder of this chapter, we will give a proof of our asynchronous time complexity theorem. We first state and prove a lemma about the protocol complex of a protocol in the IIS model with only one available IS object.

Lemma 4.3 *Let $\mathcal{A}$ be an input complex in the IIS model with a single IS object. The corresponding protocol complex is isomorphic to $\mathcal{X}(\mathcal{A})$.*

Proof. We will construct an isomorphism Ψ from the abstract complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ to the abstract complex (vertex scheme) $\mathcal{X}(A)$, as specified by Lemma 3.30. Let $\vec{v} = \langle i, S_i \rangle$ be any vertex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. Then $\Psi(\vec{v}) = \langle i, T_i \rangle$, where T_i is the simplex in $\mathcal{A}$ such that for all j, $S_i[j] = v_j$ if and only if $\langle j, v_j \rangle \in T_i$. Notice that this isomorphism is chromatic, that is, the id of a vertex equals the id of its image under Ψ.

By Lemma 3.30, we must show that a set of vertexes $\vec{v}_0, \ldots, \vec{v}_m$ in $skel^0(\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A}))$, where $m \leq n$, form a simplex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ if and only if the set of vertexes $\Psi(\vec{v}_0), \ldots, \Psi(\vec{v}_m)$

in $skel^0(\mathcal{X}(\mathcal{A}))$ form a simplex in $\mathcal{X}(\mathcal{A})$. Suppose without loss of generality that for all i, where $0 \leq i \leq m$, $\vec{v}_i = \langle i, S_i \rangle$, where $S_i \in \vartheta(D_I)$, and D_I is the input data type (that is, the id of the i-th vertex is i).

Suppose that the vertexes $\vec{v}_0, \ldots, \vec{v}_m$ do form a simplex V in in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. This output simplex corresponds to some execution α in the 1-shot IS model, with corresponding input simplex U in $\mathcal{A}$. Each vertex in U is labeled with a process id i and an input value $v_i \in D_I$. Notice that $\dim(V) \leq \dim(U)$, since some participating processes may not decide, that is, they may fail (execute a $fail_i$ action) before executing a *decide* action.

From Lemma 2.17, we have that, for any vertex $\vec{v}_i = \langle i, S_i \rangle$ in V, $S_i[i] = v_i$. This implies that $\langle i, v_i \rangle$ is in T_i. From Lemma 2.18, we have that, for any two vertexes $\vec{v}_i = \langle i, S_i \rangle$ and $\vec{v}_j = \langle j, S_j \rangle$ in V, either S_i is a prefix of S_j or vice versa. Suppose without loss of generality that S_j is a prefix of S_i. Then for all x, where $0 \leq x \leq n$, if $S_i[x] = \perp$, then $S_j[x] = \perp$, and if $S_j[x] \neq \perp$ then $S_i[x] = S_j[x]$. It follows that if $\langle x, v_x \rangle$ is in T_j it is also in T_i, and if x is not in $ids(T_i)$, then it is also not in $ids(T_j)$ This implies that T_j is a face of T_i. From Lemma 2.19, it follows that, if $S_i[j] = v_j$, then S_j is a prefix of S_i. This means that, if $\langle j, v_j \rangle$ is in T_i, then T_j is a face of T_i.

Now suppose that the vertexes $\Psi(\vec{v}_0), \ldots, \Psi(\vec{v}_m)$ in $skel^0(\mathcal{X}(\mathcal{A}))$ form a simplex V in $\mathcal{X}(\mathcal{A})$. We will construct an execution α with corresponding output simplex U such that $\Psi(U) = V$. Let $W = carrier(V)$. Partition the set $ids(V)$ into a collection of nonempty *concurrency classes* of process ids, $\mathcal{C}_1, \ldots, \mathcal{C}_k$ for some $k \geq 0$, such that any two process indices i, j are in the same concurrency class if and only if $T_i = T_j$.

We can define a total order $\prec$ on this collection of concurrency classes as follows. Let $\mathcal{C}_x, \mathcal{C}_y$ be distinct concurrency classes. Then $\mathcal{C}_x \cap \mathcal{C}_y = \emptyset$. Since both classes are nonempty, we can pick an element from each, say $i \in \mathcal{C}_x$ and $j \in \mathcal{C}_y$. By assumption, $T_i \neq T_j$. Then be Lemma 3.30, either T_i is a face of T_j or T_j is a face of T_i. In the first case, let $\mathcal{C}_x \prec \mathcal{C}_y$, and in the second case, let $\mathcal{C}_y \prec \mathcal{C}_x$. The faces of a simplex are totally ordered, and hence $\prec$ is a total order of the concurrency classes.

Now use this ordered partition of the participating processes in α to define a second partition $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$ of the set $ids(W)$ as follows. For each concurrency class $\mathcal{C}$ of $ids(V)$, define a concurrency class $\mathcal{C}'$ of $ids(W)$ as follows. $\mathcal{C}'$ is the union of $\mathcal{C}$ and all $i \in ids(W) - ids(V)$ such that $\mathcal{C}$ is the least concurrency class (as determined by $\prec$) such that for all

$j \in \mathcal{C}$, $i \in T_j$. Note that this is a partition of all of $ids(W)$ since $W = carrier(S)$. This partition gives us a new collection of concurrency classes $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$.

We are now ready to construct α. First position $update_{\mathcal{C}'_i}$ actions in increasing order according to the $\prec$ ordering. For each concurrency class $\mathcal{C}'_x$, position the *inv_writeread(v)*$_{i,IS_1}$ actions of all i such that $i \in \mathcal{C}_x$ immediately before the $update_{\mathcal{C}'_x}$ action (their internal ordering does not matter). Similarly, position the *ret_writeread(v)*$_{i,IS_1}$ and *decide(S)*$_i$ actions of all i such that $i \in \mathcal{C}_x$ and $i \in ids(V)$ immediately after the $update_{\mathcal{C}'_x}$ action, but before the *inv_writeread(v)*$_{i,IS_1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. Processes i whose index is not in $ids(W)$ do not participate and hence take no steps in α. Processes i whose index is in some concurrency class $\mathcal{C}'_x$ but not in $ids(V)$ do not execute a *ret_writeread(v)*$_{i,IS_1}$ action, instead they execute a $fail_i$ action after the $update_{\mathcal{C}'_x}$ action, but before the *inv_writeread(v)*$_{i,IS_1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. By construction, each deciding process i decides S_i in α, as required. The lemma follows. $\square$

We now consider the protocol complex of a protocol in NIIS with time complexity 1 on the input complex $\mathcal{I}$, that is, some processes access a single IS object, while some decide based only on their own inputs. We will show that, if δ is trivial, which we denote by $\delta = 1$, then this protocol complex is indeed a non-uniform chromatic subdivision.

Lemma 4.4 *The protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any NIIS protocol of time complexity 1 with input complex $\mathcal{I}$ is equal to some non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^1(\mathcal{I})$ up to isomorphism.*

Proof. We will show how to construct the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$, and prove this construction is in accordance with Definition 3.31.

Consider any vertex $\vec{v}$ in $\mathcal{I}$. It is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in Section 2.5, process i will (provided it does not fail), upon having received the input v_i, either execute an action *inv_writeread(v)*$_{i,IS_1}$ or *decide(S)*$_i$, depending on whether $\tau($*local_state*$)$ evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. Since the time complexity of $\mathcal{P}_{(n,\tau,1)}$ on any input simplex in $\mathcal{I}$ is 1, the set A must be nonempty. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.31, that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ if and only if all its vertexes are in A, and it is in $\mathcal{B}$ if and only if all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an *inv_writeread(v)*$_{i,IS_1}$ action with the object IS_1. By Lemma 4.3, the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ equals $\mathcal{X}(\mathcal{A})$ up to isomorphism. A simplex U is in $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ if and only if it corresponds to an output vector of an execution α of the protocol. In any execution α of the protocol, some of the participating, non-failing processes decide on their input values (corresponding to vertexes in B), while some decide on the snapshots they receive from the object IS_1 (corresponding to vertexes in $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$). It follows that $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ contains any simplex in $\mathcal{B}$, any simplex in $\mathcal{P}_{(n,\tau,1)}(\mathcal{A}) = \mathcal{X}(\mathcal{A})$, and any simplex of the form $S \star T$, where S is in $\mathcal{X}(\mathcal{A})$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$. The lemma follows. □

Lemma 4.5 *For all $k > 0$, the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model, with time complexity k on inputs in $\mathcal{I}$ is equal to some non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ up to isomorphism.*

Proof. We use induction on the time complexity k. By Lemma 4.4, the result holds for $k = 1$. Now suppose $k > 1$, and that the result holds for $1, \ldots, k-1$. Consider the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model with time complexity k on inputs in $\mathcal{I}$.

Any vertex $\vec{v}$ in $\mathcal{I}$ is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in Section 2.5, any non-failing process i will, upon having received the input v_i, either execute an action *inv_writeread(v)*$_{i,IS_1}$ or *decide(S)*$_i$, depending on whether $\tau(v_i)$ evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. Since the time complexity of $\mathcal{P}_{(n,\tau,1)}$ on inputs in $\mathcal{I}$ is k, the set A must be nonempty. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.31, that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ iff all its vertexes are in A, and it is in $\mathcal{B}$ iff all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an *inv_writeread(v)*$_{i,IS_1}$ action with the object IS_1. By Lemma 4.3, the output protocol complex on inputs in $\mathcal{A}$ after the first IS access equals $\mathcal{X}(\mathcal{A})$. The final protocol complex

$\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ is given by applying $\mathcal{X}(\mathcal{A})$ as an input complex to the Protocol. Since the complexity of the protocol on inputs in $\mathcal{I}$, and hence on inputs in $\mathcal{A}$, is k, the complexity of the protocol on inputs in $\mathcal{X}(\mathcal{A})$ must be $k-1$. It follows by induction that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ equals some non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ of $\mathcal{X}(\mathcal{A})$ up to isomorphism. A simplex U is in $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ iff it corresponds to a valid set of outputs of an execution α of the protocol. In any execution α of the protocol, some of the participating, non-failing processes decide on their input values, while some decide on the snapshots they receive from some IS object. It follows that $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ contains any simplex in $\mathcal{B}$, and simplex in $\mathcal{P}_{(n,\tau,1)}(\mathcal{A}) = \widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and any simplex of the form $S \star T$, where S is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$. It follows from Definition 3.31 that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ equals some non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of $\mathcal{I}$ up to isomorphism. □

We must also prove that, for any mappable non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model.

Lemma 4.6 *For any mappable non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model such that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I}) = \widetilde{\mathcal{X}}^{k}(\mathcal{I})$ up to isomorphism.*

Proof. Given a vertex $\vec{v}$ in $\mathcal{I}$. The definition of the subdivision $\widetilde{\mathcal{X}}^{k}(\mathcal{I})$ induces a sequence of non-uniform chromatic subdivisions $\mathcal{I}, \widetilde{\mathcal{X}}^{1}(\mathcal{I}), \ldots, \widetilde{\mathcal{X}}^{k}(\mathcal{I})$, and corresponding sequences $\mathcal{A}_0, \ldots, \mathcal{A}_{k-1}$ and $\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}$ of complexes, the former sequence specifying the subcomplex to be subdivided further at each level of recursion.

In order to construct a protocol for $n+1$ processes, we must specify the function $\tau : \bigcup_{l=0}^{k} \vartheta^{l}(D) \rightarrow \{\texttt{true}, \texttt{false}\}$ and the decision map $\delta : \bigcup_{l=0}^{k} \vartheta^{l}(D) \rightarrow D_O$. We specify τ to be `true` for all values v such that there is a vertex $\vec{v}$ in one of the complexes $\mathcal{A}_0, \ldots, \mathcal{A}_{k-1}$ with $val(\vec{v}) = v$, and τ to be `false` for all values v such that there is a vertex $\vec{v}$ in one of the complexes $\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}$ with $val(\vec{v}) = v$. For all other values v, τ evaluates to `false`. This definition is well-formed, since for all p, where $0 \leq p \leq k$, it follows from Definition 3.29 and Definition 3.31 that there are no two vertexes in $\widetilde{\mathcal{X}}^{p}(\mathcal{I})$ with the same process-value label pair, and for all p, q, where $0 \leq p, q \leq k$ and $p \neq q$, $\mathcal{B}_p$ and $\mathcal{B}_q$ have no vertexes with common labels (process id *and* value label). This concludes the proof. □

We now give the proof of Theorem 4.2.

Proof. (Of Theorem 4.2) Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ be a decision task. Lemma 4.5 states that any protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$, with worst case complexity k_S on input S, corresponds to a non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ with level k_S on S. Suppose now the decision map δ is not trivial. Then, if $\mathcal{P}_{(n,\tau,\delta)}$ solves $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, $\mu = \delta$ is a simplicial map from $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ to $\mathcal{O}$ that is in correspondence with Γ, so $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ is mappable.

Lemma 4.6 states that any mappable non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ with level k_S on S is equal to the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ (where δ is trivial) of a protocol in the NIIS model with worst case complexity k_S on input S. If there is a simplicial map μ from $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ to $\mathcal{O}$ that is consistent with Γ, then by setting $\delta = \mu$, we have a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ with complexity k_S on input S. The theorem follows. $\square$

Chapter 5

Applications of the Asynchronous Complexity Theorem

5.1 Approximate Agreement

As an application of Theorem 4.2, we will analyze the well-known *Approximate Agreement* task, in which each process i is given an input x_i taken from some finite subset of the real numbers, and is required to decide on some output y_i such that, for some predetermined $\epsilon > 0$, $\max_i y_i - \min_i y_i < \epsilon$, and for all i, $y_i \in [\min_i x_i, \max_i x_i]$. Aspnes and Herlihy [1] proved a lower bound that implies a worst case time complexity of $\left\lfloor \log_3 \frac{\max_i x_i - \min_i x_i}{\epsilon} \right\rfloor$ and an upper bound of $\left\lfloor \log_2 \frac{\max_i x_i - \min_i x_i}{\epsilon} \right\rfloor$ in the NIIS model[1]. In this chapter, we will show that this $\log_2$ vs. $\log_3$ gap is not simply a technical fluke. We specify the finite $n+1$-process *Approximate Agreement* task for $\epsilon > 0$ as follows:

- $I = \{[x_0, \ldots, x_n] \mid x_i \in V \cup \{\perp\}\}$, where V is a finite subset of $\mathbb{R}$.
- $O = \{[y_0, \ldots, y_n] \mid y_i \in V \cup \{\perp\}, (y_i, y_j \neq \perp) \Rightarrow |y_i - y_j| \leq \epsilon\}$.
- $\gamma = \{(\vec{I}, \vec{O}) \mid \vec{O}[i] \in [\min_i \vec{I}[i], \max_i \vec{I}[i]] \cup \{\perp\}\}$.

Theorem 5.1 *Given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving* Approximate Agreement *with complexity $\left\lfloor \log_d \frac{\max_i \vec{I}[i] - \min_i \vec{I}[i]}{\epsilon} \right\rfloor$ on any input vector $\vec{I}$, where $d = 3$ if the size of the*

[1]Although their proofs are for the read/write register model, they carry over to the NIIS model

participating set of $\vec{I}$ is 2, and $d = 2$ if the size of the participating set of $\vec{I}$ is 3 or more. Moreover, this protocol is optimal on each input vector.

Theorem 4.2 provides the lower bound directly, and the matching upper bound protocol follows from the subdivision and the simplicial map. We hope to convince the reader that this is an excellent example of how topological modeling exposes subtle points which would otherwise be difficult to grasp.

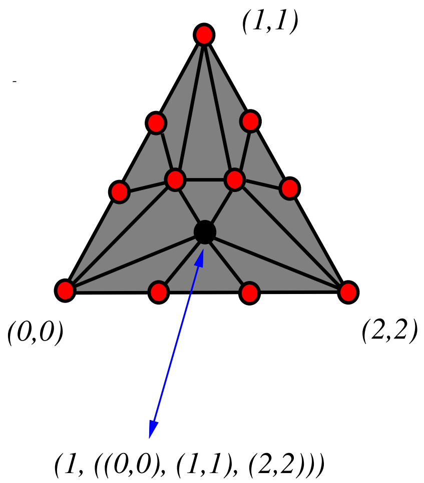

Figure 5-1: Simplex Subdivided by an *Approximate Agreement* Protocol

The key intuition behind our ability to close the gap between the upper and lower bounds for *Approximate Agreement* is depicted in Figure 5-1, which shows the subdivisions induced by a three process protocol on the input vector $[0, 1, 2]$. Aspnes and Herlihy [1] derive their lower bound for any $n+1$ process algorithm from a "bad" execution in which only the two processes with inputs farthest apart participate. Such an execution in our model corresponds to a sequence of chromatic subdivisions of the edge between $\langle 0, 0\rangle$ and $\langle 2, 2\rangle$.

In the end, the vertexes of each 1-simplex in the non-uniform chromatic subdivision of the edge connecting $\langle 0, 0\rangle$ and $\langle 2, 2\rangle$ must be mapped by a simplicial map to vertexes with output values that are less than ϵ apart, and hence connected by a simplex in the output complex. Since each subdivision introduces two new vertexes and splits the edge in three, in $\log_3 2$ such steps one can cut the distance among these vertexes to ϵ. However, note that if one considers executions in which 3 processes participate, we run into a problem: No matter how we subdivide the 2-simplex, there is always a path of 1-simplexes between $\langle 0, 0\rangle$ and

$\langle 2,2\rangle$ that includes the vertex $\langle 1,(\langle 0,0\rangle,\langle 1,1\rangle,\langle 2,2\rangle)\rangle$ (marked by a darker color). Hence, for 3 process executions, only a single new vertex will be introduced to this path between $\langle 0,0\rangle$ and $\langle 2,2\rangle$, and so the maximum distance between any two vertexes is cut by at most a half for each level of subdivision. Hence our tight $\log_2$ lower bound. Our upper bounds follow directly from Theorem 4.2 by specifying the proper subdivision and map. The key thing to note about the proof of Theorem 5.1 is that it does not involve any mention of the actual executions; All we need to do is argue about the topology of the inputs and outputs and then apply Theorem 4.2.

Proof. We first restate the description of the *Approximate Agreement* task using our topological framework as follows:

- $\mathcal{I}$ is the closure under containment of the collection of all simplexes of the form $(\langle 0,x_0\rangle,\ldots,\langle n,x_n\rangle)$, where for all i, $x_i \in V$.
- $\mathcal{O}$ is the closure under containment of the collection of all simplexes of the form $(\langle 0,y_0\rangle,\ldots,\langle n,y_n\rangle)$, where for all i,j, $y_i \in V$ and $|y_i - y_j| \leq \epsilon$.
- $\Gamma = \{(S,T) \mid vals(T) \subseteq [\min vals(S), \max vals(S)]\}$.

Note that the size of the participating set for the input vector corresponding to a simplex S in $\mathcal{I}$ equals $\dim(S)+1$. Theorem 5.1 then states that, given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving *Approximate Agreement* with complexity $\left\lfloor \log_d \frac{\max vals(S) - \min vals(S)}{\epsilon} \right\rfloor$ on any input simplex$S$, where $d=3$ if $\dim(S)=1$, and $d=2$ if $\dim(S)\geq 1$. Moreover, this protocol is optimal on each input simplex S.

We first establish the lower bound. Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol that solves *Approximate Agreement* with worst case complexity k_S on S, where S is any input simplex of dimension $n \geq \dim(S) > 0$. Let $D(S) = \max_{\vec{v},\vec{u}\in S} val(\vec{v}) - val(\vec{u})$. Then Theorem 4.2 states that there is some mappable non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$, with level k_S on S. We will show that $k_S \geq \left\lfloor \log_d \frac{D(S)}{\epsilon} \right\rfloor$. The proof uses the following lemma.

Lemma 5.2 *Let $l \leq k$. Label the vertexes of $\widetilde{\mathcal{X}}^l(S)$ with real numbers in a way that agrees with the initial value labeling of S, and let l_S be the level of $\widetilde{\mathcal{X}}^l(S)$. Then*

$$D(\tilde{\mathcal{X}}^l(S)) \geq \frac{D(S)}{d^{l_S}}$$

Proof. Suppose without loss of generality that $l = l_S$. We first give the proof for the case of two participating processes and $d = 3$. By definition of $D(S)$, there is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\mathcal{S}$ such that $D(U) = D(S)$. The complex $\tilde{\mathcal{X}}^l(U)$ contains at most 3^l 1-simplexes, denoted $U_1, \ldots, U_M$, where $M \leq 3^l$. These form a continuous path from $\vec{u}_0$ to $\vec{u}_1$, the endpoints of which are labeled with $val(\vec{u}_0)$ and $val(\vec{u}_1)$, respectively. So the best we can do is cut $D(U)$ in 3^l pieces. The triangle inequality tells us that $D(U) \leq \sum_{i=1}^{M} D(U_i) \leq M \max_i D(U_i) \leq 3^k \max_i D(U_i)$. Hence $\max_i D(U_i) \geq D(U)/3^l = D(S)/3^l$. The lemma follows, since $\max_i D(U_i) \leq D(\tilde{\mathcal{X}}^l(S))$.

We now prove the case where the size of the participating set is greater than 2 (and hence $\dim(S)$ is greater than 1) and $d = 2$. We argue by induction on l. The case $l = 0$ is trivial. Now suppose the claim is true for $l - 1$. By definition of $D(\tilde{\mathcal{X}}^{l-1}(S))$, there is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\tilde{\mathcal{X}}^{l-1}(S)$ such that $D(U) = D(\tilde{\mathcal{X}}^{l-1}(S))$. U is a face of some 2-simplex $U' = (\vec{u_0}, \vec{u_1}, \vec{u}_2)$. Suppose first that the next level of non-uniform chromatic subdivision does not subdivide U completely. Then there is some 1-simplex T in the level l non-uniform subdivision of U' with $D(T) \geq D(U)/2$. Since $D(U) = D(\tilde{\mathcal{X}}^{l-1}(S))$ and $D(T) \leq D(\tilde{\mathcal{X}}^l(S))$, the lemma follows by induction. Suppose instead that the next level of subdivision does subdivide U' completely. Then the level l subdivision has an internal vertex $\vec{m}_2$, colored with $id(\vec{u}_2)$, and two neighboring 1-simplexes $T_0 = (\vec{u}_0, \vec{m}_2)$ and $T_1 = (\vec{m}_2, \vec{u}_1)$. The triangle inequality then tells us that $D(U) \leq D(T_0) + D(T_1) \leq 2 \max_i D(T_i)$, where $i \in \{0, 1\}$. It follows that $D(\tilde{\mathcal{X}}^l(S)) \geq D(\tilde{\mathcal{X}}^{l-1}(S))/2$. The lemma follows by induction. □

Suppose now that there exists a chromatic simplicial map $\mu : \tilde{\mathcal{X}}^k(\mathcal{I}) \to \mathcal{O}$ such that, for all simplexes T in $\tilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T) \in \Gamma(carrier(T))$. We can associate this map with a labeling of the vertexes in $\tilde{\mathcal{X}}^k(\mathcal{I})$ as follows. Label each vertex $\vec{v}$ in $\tilde{\mathcal{X}}^k(\mathcal{I})$ with $val(\mu(\vec{v}))$. This labeling agrees with the input value labeling of $\mathcal{I}$, since for any vertex $\vec{v}$, the task specification requires that for any simplex S_0 that contains $\vec{v}$, it must be the case that $\mu(\vec{v}) \in D(S_0)$. Choose two neighboring simplexes S_0 and S_1 containing $\vec{v}$ such that $D(S_0) \cap D(S_1) = val(\vec{v})$. It follows that $\mu(\vec{v}) = val(\vec{v})$. Now let T be any simplex in $\tilde{\mathcal{X}}^k(\mathcal{I})$. By definition of μ, $\mu(T)$ is a simplex in $\mathcal{O}$, and hence $D(\mu(T)) < \epsilon$. It follows that $D(T) = D(\mu(T)) < \epsilon$, and hence that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$, where $D(\widetilde{\mathcal{X}}^k(\mathcal{I}))$ is equal to $max_{T \in \widetilde{\mathcal{X}}^k(\mathcal{I})} D(T)$. Clearly, for any input simplex S, it follows that the labels on the restriction of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ to S, $\widetilde{\mathcal{X}}^k(S)$, have range less than ϵ. The previous lemma then states that $\epsilon > D(\widetilde{\mathcal{X}}^k(S)) \geq \frac{D(S)}{d^{k_S}}$. We conclude that

$$k_S \geq \left\lfloor \log_d \frac{D(S)}{\epsilon} \right\rfloor$$

To prove the upper bound, we construct a mappable non-uniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of the input complex with level $k_S = \left\lfloor \log_d \frac{D(S)}{\epsilon} \right\rfloor$ on each input simplex S, according to Definition 3.31. As argued above, the requirement that the subdivision be mappable is equivalent to saying that there is a vertex labeling of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ that agrees with the initial value labeling of $\mathcal{I}$ with the additional property that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$.

For each level of subdivision $l \leq k$, a vertex $\vec{v}$ is in A if there is another vertex $\vec{u}$ such that $val(\vec{v}) - val(\vec{u}) > \epsilon$, otherwise it is in B. Before applying the next level of subdivision to $\mathcal{X}(\mathcal{A})$ (as specified by Definition 3.31), we relabel all new vertexes in $\mathcal{X}(\mathcal{A})$ (those not in $skel^0(\mathcal{A})$) as follows: If the dimension of $\mathcal{A}$ is 1, label the new vertexes in $\mathcal{X}(\mathcal{A})$ with $(2 \min val(\mathcal{A}) + \max val(\mathcal{A}))/3$ and $(\min val(\mathcal{A}) + 2 \max val(\mathcal{A}))/3$, respectively. This cuts the distance between the vertexes with values apart in 3. Otherwise, label the new vertexes with $(\min val(\mathcal{A}) + \max val(\mathcal{A}))/2$. This cuts the distance between the values furthest apart in 2.

It is clear from this construction that, at each level of recursion, for all simplexes S in $\mathcal{I}$ we have that, if $D(\widetilde{\mathcal{X}}^l(S)) > \epsilon$, then either $D(\widetilde{\mathcal{X}}^{l+1}(S)) = D(\widetilde{\mathcal{X}}^l(S))/d$, or $D(\widetilde{\mathcal{X}}^{l+1}(S)) < \epsilon$. It follows that the level k_S of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ on S is $\left\lfloor \log_d \frac{D(S)}{\epsilon} \right\rfloor$, where $d = 3$ if $\dim(S) = 1$, and $d = 2$ if $\dim(S) > 1$. We conclude from Theorem 4.2 that there is a wait-free protocol that solves *Approximate Agreement* with worst case time complexity $\left\lfloor \log_d \frac{D(S)}{\epsilon} \right\rfloor$ on input S, where $d = 3$ for two participating processes and $d = 2$ for three or more. $\square$

Chapter 6

Conclusion and Directions for Further Research

In this thesis, we have extended the topological framework of Herlihy and Shavit [21, 22, 26] to obtain a complete characterization of the complexity of solving decision tasks in the NIIS model, a generalization of Borowsky and Gafni's IIS model [10]. The main difference between Theorem 4.2 and Herlihy and Shavit's Asynchronous Complexity Theorem is that in our proof, we construct an explicit protocol complex for the NIIS model, and show that this complex is indeed equal to a non-uniform chromatic subdivision. Since non-uniform chromatic subdivisions have a recursive structure, they are well-suited for arguing about complexity - according to Theorem 4.2, the level of recursion of a mappable non-uniform chromatic subdivision of a task's input complex *is* the complexity of the corresponding wait-free NIIS solution protocol.

We have applied Theorem 4.2 to tighten the upper and lower bounds on solving the *Approximate Agreement* task implied by the work of Aspnes and Herlihy [1], by proving matching upper and lower bounds of $\left\lfloor \log_d \frac{input-range}{\epsilon} \right\rfloor$ where $d = 3$ for two processes and $d = 2$ for three or more. The intuition behind this result, as well as its formal proof, is based on simple, geometric and topological arguments about the level of non-uniform chromatic subdivision that is necessary and sufficient for mappability. We believe this is an excellent example of how Theorem 4.2 exposes subtle properties of protocols in asynchronous shared memory systems, and how it allows us to reason formally about them without having to argue directly about concurrent executions.

A possible direction for further research is to apply Theorem 4.2 to other decision tasks, such as for example *Renaming*. A good first step towards this end would be to formulate a precise statement of the task using our topological framework, and then state and prove a version of Theorem 4.2 for comparison-based protocols, which are guaranteed to satisfy the symmetry requirements of the *Renaming* task.

Another possible direction is to try to extend our topological framework to other models of computation, such as the atomic snapshot model, the single-writer multi-reader model, or even the multi-writer multi-reader model. Our choice of the NIIS model was motivated by the fact that its protocol complex is highly structured, and corresponds to a non-uniform chromatic subdivision, as the proof of Theorem 4.2 shows. Other, less restricted models, such as the ones mentioned above, do not have this property, and so in order to prove a result similar to Theorem 4.2 in any of these models, one would need to identify some invariant, recursive substructure that one can model topologically with reasonable ease.

An alternative approach would be to use simulation techniques to relate the NIIS model to other models of computation, thereby obtaining an indirect characterization of the complexity of solving decision tasks in these models. Currently, however, the best known wait-free simulation of a single IS object using atomic snapshots requires $O(N)$ accesses to shared memory by each process, where N is the number of processes. There is thus an important open problem in finding an *optimal*, wait-free implementation of NIIS using atomic snapshot, and vice versa.

Finally, it is possible to extend the work presented in this thesis by using our existing topological framework to develop a characterization of the *work complexity* involved in solving decision tasks in the NIIS model. As was the case for time complexity, a mappable non-uniform chromatic subdivision of an input complex does contain the information necessary to describe work complexity. The main difficulty to be overcome is to find an easy way of extracting this information from the subdivided complex. One possible approach to this problem would involve some form of topological invariant that keeps track of the maximal *sum* of the number of IS objects accessed by the processes corresponding to the vertexes in each simplex in the subdivision.

Bibliography

[1] J. Aspnes and M.P. Herlihy. Wait-free data structures in the asynchronous pram model. In preparation, 1996.

[2] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 377–408, August 1990.

[3] H. Attiya, N. Lynch, and N. Shavit. Are wait-free algorithms fast? In *Proceedings Of The 31st Annual Symposium On The Foundations of Computer Science*, October 1990.

[4] H. Attiya and S. Rajsbaum. A combinatorial topology framework for wait-free computability. Preprint, 1995.

[5] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rudiger Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, July 1990.

[6] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 263–275, August 1988.

[7] E. Borowski. Capturing the power of reseiliency and set consensus in distributed systems. Technical report, University of California Los Angeles, Los Angeles, California, 1995.

[8] E. Borowsky and E. Gafni. Generalized flp impossibility result for t-resilient asynchronous computations. In *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993.

[9] E. Borowsky and E. Gafni. Immediate atomic snapshots and fast renaming. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, August 1993.

[10] E. Borowsky and E. Gafni. A simple algorithmically reasoned chracterization of wait-free computationsi. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, August 1997.

[11] S. Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings Of The Ninth Annual ACM Symosium On Principles of Distributed Computing*, pages 311–234, August 1990.

[12] S. Chaudhuri, M.P. Herlihy, N. Lynch, and M.R. Tuttle. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, October 1993.

[13] D. Dolev, N.A. Lynch, S.S. Pinter, E.W. Stark, and W.E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[14] A. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, August 1986.

[15] M. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed commit with one faulty process. *Journal of the ACM*, 32(2), April 1985.

[16] E. Gafni and E. Koutsoupias. 3-processor tasks are undecidable. PODC 95, Brief announcement.

[17] J. Havlicek. Computable obstructions to wait-free computability. In *Proceedings of the 1997 ACM Symposium on Theory of Computing*, pages –, 1997.

[18] M. Herlihy and S. Rajsbaum. Algebraic spans. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 90–99. ACM, August 1995.

[19] M.P. Herlihy and S. Rajsbaum. On the decidability of distributed decision problems. In preparation.

[20] M.P. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, August 1994.

[21] M.P. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993.

[22] M.P. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of the 1994 ACM Symposium on Theory of Computing*, May 1994.

[23] N. Lynch and S. Rajsbaum. On the borowsky-gafni simulation algorithm. In *ISTCS Israel Symposium on Theory of Computing and Systems*, 1996.

[24] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. Technical Report MIT/LCS/TM-373, MIT Laboratory For Computer Science, November 1988.

[25] Nancy A. Lynch. *Distributed Algorithms.* Morgan-Kaufman, San Francisco, CA, 1996. ISBN 1-55860-348-4.

[26] N. Shavit M.P. Herlihy. The topological structure of asynchronous computability. 1995.

[27] J.R. Munkres. *Elements Of Algebraic Topology.* Addison Wesley, Reading MA, 1984. ISBN 0-201-04586-9.

[28] G. Neiger. Set-linearizability. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, page 396. ACM, August 1994.

[29] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993.

[30] E. Schenk. Computability and complexity results for agreement problems in shared memory distributed systems. Technical report, University of Toronto, Toronto, Canada, 1996.

[31] E.H. Spanier. *Algebraic Topology.* Springer-Verlag, New York, 1966.