Department of Computer Science

Logic and Complexity in Distributed Computing

Tuomo Lempiäinen

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 55/2019

Logic and Complexity in Distributed Computing

Tuomo Lempiäinen

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall M1 of the school on 4 April 2019 at 12 noon.

Aalto University
School of Science
Department of Computer Science

Supervising professor
Professor Jukka Suomela, Aalto University, Finland

Thesis advisor
Professor Jukka Suomela, Aalto University, Finland

Preliminary examiners
Professor Olivier Carton, University Paris Diderot, France
Professor Yuval Emek, Technion, Israel

Opponent
Professor Yuval Emek, Technion, Israel

Aalto University publication series
DOCTORAL DISSERTATIONS 55/2019

ISBN 978-952-60-8477-0 (printed)
ISBN 978-952-60-8478-7 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-8478-7

Unigrafia Oy
Helsinki 2019

Finland

NORDIC SWAN ECOLABEL
Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Tuomo Lempiäinen

Name of the doctoral dissertation
Logic and Complexity in Distributed Computing

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 55/2019

Field of research Information and Computer Science

Manuscript submitted 13 November 2018 **Date of the defence** 4 April 2019

Permission to publish granted (date) 15 January 2019 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

This dissertation studies the theory of distributed computing. In the distributed setting, computation is carried out by multiple independent computational units. They communicate with neighbouring units and collectively solve a problem. Systems of this kind are widespread in the information society as well as in the natural world: prime examples include the Internet, multiprocessor computer systems, the cells of a biological organism and human social networks. Therefore it is important to gain knowledge on the fundamental capabilities and limitations of distributed systems.

Our work takes place in the context of deterministic and synchronous message-passing models of distributed computing. Such models abstract away some possible features of distributed systems, such as asynchrony, congestion and faults, while putting emphasis on the amount of communication needed between the units. The computational units are expected to solve a problem related to the structure of the underlying communication network. More specifically, we study the standard LOCAL model, the standard port-numbering model and several weaker variants of the latter.

The contributions of this dissertation are two-fold. First, we give new methods for studying distributed computing by establishing a strong connection between several weak variants of the port-numbering model and corresponding variants of modal logic. This makes it possible to apply existing logical tools, such as bisimulation, to understand computation in the distributed setting. We also give a full characterisation of the relationships between the models of computing, which has implications on the side of mathematical logic.

Second, we prove the existence of various new non-empty complexity classes. One of our results studies the relationship between time and space complexity, which is a relatively new topic in distributed computing research. We demonstrate the existence of a problem that can be solved in a constant amount of space in a very weak model but nonetheless requires a non-constant amount of time even in a much stronger model of computing. Another result continues the recent success on developing complexity theory for the LOCAL model. We introduce a new technique which shows the existence of an infinite hierarchy of problems with novel time complexities.

Keywords distributed computing, computational complexity theory, port-numbering model, LOCAL model, graph problems, LCL problems, modal logic, bisimulation

ISBN (printed) 978-952-60-8477-0 **ISBN (pdf)** 978-952-60-8478-7

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2019

Pages 140 **urn** http://urn.fi/URN:ISBN:978-952-60-8478-7

Tekijä
Tuomo Lempiäinen

Väitöskirjan nimi
Logiikka ja vaativuus hajautetussa laskennassa

Julkaisija Perustieteiden korkeakoulu

Yksikkö Tietotekniikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 55/2019

Tutkimusala Tietojenkäsittelytiede

Käsikirjoituksen pvm 13.11.2018 **Väitöspäivä** 04.04.2019

Julkaisuluvan myöntämispäivä 15.01.2019 **Kieli** Englanti

☐ **Monografia** ☒ **Artikkeliväitöskirja** ☐ **Esseeväitöskirja**

Tiivistelmä

Tässä väitöskirjassa tutkitaan hajautetun laskennan teoriaa. Hajautetussa tilanteessa laskentaa suorittavat useat itsenäiset laskentayksiköt. Ne kommunikoivat viereisten laskentayksiköiden kanssa ja yhdessä tuottavat ratkaisun johonkin ongelmaan. Tällaisia järjestelmiä esiintyy laajalti niin tietoyhteiskunnassa kuin luonnossakin: hyviä esimerkkejä ovat Internet, usean suorittimen tietokonejärjestelmät, biologisen organismin solut tai vaikkapa ihmisten muodostamat sosiaaliset verkostot. Siksi on tärkeää saada tietoa tällaisten järjestelmien perustavanlaatuisista kyvyistä ja rajoituksista.

Tässä työssä keskitytään hajautettuun laskentaan deterministissä ja synkronisissa viestinvälitysmalleissa. Tällaiset mallit jättävät huomiotta jotkin mahdolliset hajautettujen järjestelmien ominaisuudet, esimerkiksi asynkronian, ruuhkautumisen ja vikaantumisen, ja niiden sijaan korostavat tarvittavaa laskentayksiköiden välisen kommunikaation määrää. Yksiköiden odotetaan ratkaisevan ongelman, joka koskee järjestelmän pohjalla olevan kommunikaatioverkon rakennetta. Tarkalleen ottaen työssä tutkitaan paljon tutkittuja LOCAL- ja porttinumerointimalleja sekä useita viimeksi mainitun mallin heikompia variantteja.

Väitöskirjalla on kaksi päätulosta. Ensiksi työssä esitellään uusia menetelmiä hajautetun laskennan tutkimiseen todistamalla vahva yhteys useiden porttinumerointimallin heikkojen varianttien ja vastaavien modaalilogiikan varianttien välille. Tämä mahdollistaa olemassaolevien logiikan välineiden, esimerkiksi bisimulaation, soveltamisen hajautetun laskennan ymmärtämiseen. Lisäksi työssä karakterisoidaan täydellisesti näiden laskennan mallien väliset suhteet, millä on seurauksia myös matemaattisen logiikan alueella.

Toiseksi väitöskirjassa osoitetaan useiden uusien epätyhjien vaativuusluokkien olemassaolo. Yhdessä tuloksista tutkitaan aika- ja tilavaativuuden välistä suhdetta, joka on varsin uusi aihe hajautetun laskennan tutkimuksessa. Tulos osoittaa, että on olemassa ongelma, joka voidaan ratkaista vakiotilassa hyvin heikossa laskennan mallissa mutta joka vaatii vakiota suuremman määrän aikaa, vaikka laskennan malli olisi paljon vahvempi. Toinen tulos jatkaa viimeaikaista menestyksekästä vaativuusteorian kehittämistä LOCAL-mallille. Työssä esitellään uusi tekniikka, jonka avulla voidaan muodostaa ääretön hierarkia ongelmia, jotka kuuluvat uusiin aikavaativuusluokkiin.

Avainsanat hajautettu laskenta, laskennan vaativuusteoria, porttinumerointimalli, LOCAL-malli, verkko-ongelmat, LCL-ongelmat, modaalilogiikka, bisimulaatio

ISBN (painettu) 978-952-60-8477-0 **ISBN (pdf)** 978-952-60-8478-7

ISSN (painettu) 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Helsinki **Painopaikka** Helsinki **Vuosi** 2019

Sivumäärä 140 **urn** http://urn.fi/URN:ISBN:978-952-60-8478-7

Preface

The research presented in this dissertation began a while ago – first at the Department of Information and Computer Science of Aalto University. After working for a while at the University of Helsinki and also obtaining my master's degree in mathematics there, I returned to Aalto University in 2015 to pursue my doctoral studies. For the last few years, I worked under the auspices of the grand unified Department of Computer Science.

Departments aside, I would like to first and foremost thank my supervisor Jukka Suomela for providing me with an opportunity to conduct my doctoral research in his first-class research group and for his guidance and patience over the years. I have learned a lot from him – on the theory of distributed computing, on how to conduct research and on how to survive in the world of research.

My other co-authors, of whom there are plenty, also played a very important role in my research output. Many thanks to Alkida Balliu, Lauri Hella, Juho Hirvonen, Matti Järvisalo, Janne H. Korhonen, Antti Kuusisto, Juhana Laurinharju, Kerkko Luosto, Dennis Olivetti and Jonni Virtema for collaborating with me on the articles included in this dissertation.

I have also been involved in projects that are not part of this dissertation. I would like to thank Sebastian Brandt, Orr Fischer, Barbara Keller, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jara Uitto and Przemysław Uznański for fruitful collaboration on distributed computing research. Furthermore, with regard to my earlier adventures in the area of DNA self-assembly, I wish to thank Eugen Czeizler, Mika Göös and Pekka Orponen.

I took my first steps as a researcher in 2010 under the guidance of Pekka Orponen – many thanks for that. I have had the privilege to interact with many excellent researches in the course of my university career. To give one example, Petteri Kaski has always been there, although I never worked for him directly.

The logic communities of Helsinki and Tampere have provided refreshing variation to the research themes surrounding me. Thanks to everyone involved, with a special mention to Juha Kontinen, to name a person not yet in my list of co-authors. I also want to give special thanks to Antti Kuusisto for hosting me during my visit to the University of Bremen in 2015.

I wish to thank Yuval Emek and Olivier Carton for pre-examining this disser-

tation. In addition, I express my thanks to Yuval Emek for also agreeing to act as the opponent in my public examination.

Financially, the work in this dissertation was mostly supported by the Aalto University Department of Computer Science. I also acknowledge the financial support of the Nokia Foundation, the Helsinki Doctoral Education Network in Information and Communications Technology, the Magnus Ehrnrooth Foundation and the Helsinki Doctoral Programme in Computer Science.

Last but not least, I would like to express my gratitude to my family, in particular my parents Anna-Maija and Olavi, as well as everyone else who has encouraged me during my studies. Finally, I want to thank Jenni for everything, including (but definitely not limited to) helping me improve the language of this dissertation.

Helsinki, 5th March 2019,

Tuomo Lempiäinen

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, volume 28, issue 1, pages 31–53, February 2015.

II Tuomo Lempiäinen. Ability to count messages is worth $\Theta(\Delta)$ rounds in distributed computing. In *Proc. 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, New York, NY, USA, pages 357–366, July 2016.

III Tuomo Lempiäinen and Jukka Suomela. Constant space and non-constant time in distributed computing. In *Proc. 21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, Lisbon, Portugal, pages 30:1–30:16, March 2018.

IV Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti and Jukka Suomela. New classes of distributed time complexity. In *Proc. 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, Los Angeles, CA, USA, pages 1307–1318, June 2018.

Author's Contribution

Publication I: "Weak models of distributed computing, with connections to modal logic"

The author devised and wrote the original proof of the central technical result of the article (Theorem 2) and participated in writing the article.

Publication II: "Ability to count messages is worth $\Theta(\Delta)$ rounds in distributed computing"

The author did everything.

Publication III: "Constant space and non-constant time in distributed computing"

The author formalised the original proof idea of Suomela and wrote the majority of the article.

Publication IV: "New classes of distributed time complexity"

The research work and write-up are joint work, with the author being the responsible person for Section 2 of the article.

1. Introduction

The world is full of distributed systems, both large and small. They can be found in nature and in various computer and communication systems, as well as in human societies. Therefore it is important to be able to reason about them: what kind of operations they are able to carry out and how efficiently – and what is impossible to them. The aim of this dissertation is to advance our understanding of the theory behind distributed systems in several ways: by providing new tools and connections to other fields, as well as by proving structural results about different problem classes in the distributed setting.

1.1 The distributed setting

A distributed system consists of several independent agents that communicate with each other. The agents are usually assumed to be identical with each other. The aim is to solve a given task so that each agent computes its own part of the result.

There exists a plenitude of different ways to define the details of this setting. The agents can be deterministic or randomised, they can have different amounts of memory or computational power, and the communication with other agents can take various forms.

In this dissertation, we study a setting where each agent is a deterministic computational unit, and the communication takes place in synchronous communication rounds. We define the models of computation properly in Section 2.3, after introducing some preliminary concepts.

1.2 Objectives and dissertation structure

This dissertation consists of an overview and four published articles. Our aim is to give (at least partial) answers to the following three questions:

(1) Can mathematical logic be used to understand distributed systems?

(2) What is the relationship between different resources (specifically time and space) in distributed computing?

(3) What kind of time complexity classes exist in the distributed setting?

First, we go over the most central background theory of this work in Chapter 2. Then, we discuss Publications I and II, which are related to question (1) above, in Chapter 3. Next, we present Publication III, related to question (2), in Chapter 4. After that, Publication IV, which is related to question (3), is discussed in Chapter 5. Finally, we consider the implications of this work and possible future research directions in Chapter 6.

The reader is assumed to be familiar with at least the basics of discrete mathematics (such as notions related to sets and functions), classical propositional logic as well as fundamentals of computational complexity theory, in particular the asymptotic notation.

1.3 An overview of the results

In this section, we give a short, informal overview of the key results of this dissertation. Formal definitions of the problems and detailed statements of the results can be found in the subsequent sections.

Key result 1. We define six restricted variants of the widely-studied *port-numbering model* of Angluin [2]. We denote the class of graph problems solvable in the port-numbering model by $\mathsf{VV_c}$. The following problem classes and corresponding model variants are studied in our work:

$\mathsf{VV_c}$: Input and output ports are numbered consistently.
VV: Input and output ports are numbered, but not necessarily consistently.
MV: Output ports are numbered; nodes receive a multiset of messages.
SV: Output ports are numbered; nodes receive a set of messages.
VB: Input ports are numbered; nodes broadcast one message to all neighbours.
MB: Combination of the restrictions of MV and VB.
SB: Combination of the restrictions of SV and VB.

We develop *descriptive complexity theory* for distributed computing by showing that each of the above classes is captured by a variant of *modal logic* – either *basic modal logic*, *multimodal logic*, *graded modal logic* or *graded multimodal logic* – in a suitable class of structures.

Key result 2. Some of the classes defined above seem orthogonal at first sight. However, we prove, partially by making use of the connection to modal logic, that the classes actually form a *linear order*:

$$\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}.$$

We also establish *tight upper and lower bounds* for simulating the weaker SV model in the stronger MV model.

Key result 3. We study *constant-space* computation in a distributed setting. In the centralised setting, notions such as finite automata and various space-complexity classes are well understood and can be related to time complexity. In distributed computing, this is not yet the case.

We identify a setting where it is not clear at all whether any non-trivial graph problem can be solved by using only a constant amount of memory. It turns out that even for a weak variant of the port-numbering model and cycle graphs, there exists a graph problem that can be solved in *constant space*, but whose time complexity is *linear* in the size of the graph.

Key result 4. Recently, the development of complexity theory for the standard LOCAL [28, 31] model has progressed rapidly. We take part in this progress by showing that there exist graph problems with a large variety of *previously unknown time complexities*. We obtain, for instance, the following complexities (with α being a positive rational number):

- $\Theta(\log^{\alpha} n)$ for any $\alpha \geq 1$,
- $2^{\Theta(\log^{\alpha} n)}$ for any $\alpha \leq 1$
- $\Theta(n^{\alpha})$ for any $\alpha < 1/2$,
- $\Theta(\log^{\alpha} \log^{*} n)$ for any $\alpha \geq 1$,
- $2^{\Theta(\log^{\alpha} \log^{*} n)}$ for any $\alpha \leq 1$,
- $\Theta((\log^{*} n)^{\alpha})$ for any $\alpha \leq 1$.

In particular, our results refute previously conjectured gaps in the time complexity hierarchy of the LOCAL model.

2. Preliminaries

This chapter introduces the notation and the basic definitions on which we build the work presented in the later chapters. We define the most important models of computation – the port-numbering model and the LOCAL model – as well as the basic modal logic.

2.1 Basic notation

The set of natural numbers is $\mathbb{N} = \{0, 1, \ldots\}$. Given $k, k' \in \mathbb{N}$, we write $[k]$ to denote the set $\{1, 2, \ldots, k\}$ and $[k, k']$ to denote the set $\{k, k+1, \ldots, k'\}$. The power set, that is, the set of all subsets, of a set A is denoted by $\mathscr{P}(A) = \{B : B \subseteq A\}$.

2.2 Graph theory

Let us now define the most central mathematical objects in our field: graphs. An *undirected graph* is a pair $G = (V, E)$, consisting of a set of *vertices* or *nodes* V and a set of *edges* E. Each $e \in E$ is an unordered pair $e = \{u, v\}$ for some vertices $u, v \in V$. On the other hand, in a *directed graph* $G = (V, E)$, the edges $e \in E$ are ordered pairs $e = (u, v)$ of vertices.

Let $G = (V, E)$ be an undirected graph. The *degree* of a vertex $u \in V$ is $\deg_G(u) = |\{v \in V : \{u, v\} \in E\}|$, that is, the number of *neighbours* the vertex u has in G. If $\deg_G(u) = d$ for all $u \in V$, graph G is said to be *d-regular*. A *path* P from a vertex u to a vertex v is a sequence of vertices $P = (v_1, v_2, \ldots, v_k)$ such that $v_1 = u$, $v_k = v$ and $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, 2, \ldots, k-1\}$. The *length* of the path P is defined to be $k - 1$, that is, the number of edges between the vertices. The *distance* $\operatorname{dist}_G(u, v)$ between vertices u and v of G is the length of the shortest path from u to v. These concepts are defined analogously for directed graphs.

Given an undirected graph $G = (V, E)$, we denote the set of all directed edges between vertices that are adjacent in graph G by $\vec{E} = \{(u, v) : \{u, v\} \in E\}$. In this work, a *labelled graph* is a pair (G, ℓ), where $G = (V, E)$ is an undirected graph and $\ell : V \cup \vec{E} \to \Sigma$ is a function that assigns a label $\ell(u) \in \Sigma$ to each node

$u \in V$ and a label $\ell(u,v) \in \Sigma$ to each pair $(u,v) \in \vec{E}$. An *isomorphism* between labelled graphs (G,ℓ) and (G',ℓ') is a bijection $f : V \to V'$ such that (1) for all $u,v \in V$, we have $\{u,v\} \in E$ if and only if $\{f(u),f(v)\} \in E'$, (2) for all $u \in V$, we have $\ell(u) = \ell'(f(u))$, and (3) for all $(u,v) \in \vec{E}$, we have $\ell(u,v) = \ell'(f(u),f(v))$.

We say that a graph is *simple* if there are no self-loops from a vertex to itself. If a graph contains a path from each vertex to any other vertex, it is called *connected*. In this work, all graphs are assumed to be simple, connected, finite and undirected, unless stated otherwise. Occasionally, we will make use of directed graphs. For $\Delta \in \mathbb{N}$, we denote by $\mathscr{F}(\Delta)$ the family of all (simple, connected, finite and undirected) graphs that have a maximum degree of at most Δ.

2.3 Models of computation

In this section, we introduce formally the models of distributed computation that we study in this dissertation. While there exists a huge variety of different models in the literature, the focus of this work is on models that share the following important properties: each node runs the same *deterministic* algorithm, and communication between nodes happens in *synchronous* communication rounds. Moreover, local computation inside each node is considered free, and the size of the messages is unlimited.

In our framework, distributed computing is essentially about *communication* instead of *computation* in the traditional sense. This is reflected in the main *complexity measure* we use: the number of communication rounds until all nodes have produced their outputs. This is called *time complexity*. In addition, we consider *space complexity*: the maximum amount of memory used by any node during the execution.

To give the intuition, let $G = (V,E)$ be the communication graph. A distributed algorithm $\mathscr{A}$ is executed on G as follows. In the beginning, each node $u \in V$ knows its own degree $\deg_G(u)$ and possibly a local input given to it. Then, each node u performs three operations on each communication round:

(1) send a message to each neighbour v,
(2) receive a message from each neighbour v,
(3) update internal state based on the received messages.

Each node is expected to eventually stop changing its state and produce an output. The local outputs together define a solution to a *graph problem*, where the problem instance is the communication graph G.

This setting needs to be further refined by defining the way nodes can distinguish themselves from other nodes and tell their neighbours apart. This gives rise to several different model variants. After defining graph problems in Section 2.3.1 and a general state machine framework in Section 2.3.2, we will proceed to give detailed definitions of two different well-established models of

computation in Sections 2.3.3 and 2.3.4. The models considered in our research are either one of these models or restricted versions of them defined later.

While we define the models of computation in a rigorous manner in this section, we will mostly resort to a more high-level description of algorithms in Chapters 3–5 – the idea being that with a serious amount of effort, each algorithm could be defined as a state machine in the sense of this section.

2.3.1 Graph problems

Let Σ and Γ be sets of local input and output labels, respectively. An *input labelling* for a graph $G = (V, E)$ is a mapping $i \colon V \cup \vec{E} \to \Sigma$. That is, mapping i gives an input label to each node and to both endpoints of each edge. A *solution* for (G, i) is a mapping $S \colon V \to \Gamma$ that maps each node v to its desired local output $S(v)$.

A *graph problem* is now a function $\Pi_{\Sigma,\Gamma}$ that maps each graph $G = (V, E)$ and each input labelling i for G to a non-empty set $\Pi_{\Sigma,\Gamma}(G, i)$ of solutions for G. We require graph problems to be invariant under isomorphisms: if $f \colon V \to V'$ is an isomorphism between (G, i) and (G', i'), and $S \in \Pi_{\Sigma,\Gamma}(G, i)$, then $S \circ f^{-1} \colon V' \to \Gamma$ is a solution in $\Pi_{\Sigma,\Gamma}(G', i')$. Problems without any local inputs can be modelled by setting $\Sigma = \{\emptyset\}$. When only node inputs are used, we assume that $i(u, v) = \emptyset$ for each $(u, v) \in \vec{E}$ and write simply $i \colon V \to \Sigma$. If Σ and Γ are clear from the context, we denote the problem $\Pi_{\Sigma,\Gamma}$ simply by Π.

A *decision graph problem* is a graph problem $\Pi_{\Sigma,\Gamma}$ where $\Gamma = \{\mathsf{yes}, \mathsf{no}\}$ and each instance is either a *yes-instance* or a *no-instance*. A yes-instance (G, i) is such that $\Pi_{\Sigma,\Gamma}(G, i) = \{S\}$ where $S(v) = \mathsf{yes}$ for all $v \in V$. On the other hand, a no-instance (G, i) is such that $\Pi_{\Sigma,\Gamma}(G, i) = \{S : S(v) = \mathsf{no} \text{ for some } v \in V\}$. That is, all nodes have to accept a yes-instance, while at least one node has to reject a no-instance.

2.3.2 Algorithms as state machines

To define distributed algorithms formally, we start by introducing the concept of port numbers that allows nodes to locally distinguish their neighbours from each other. Next, we give a general definition of state machines that is behind all the models of distributed computing considered in this work. Then, we explain what it means to run – or execute – a state machine on a graph. We conclude this section by considering complexity measures and outputs of algorithms.

Port numbers. A *port* of a graph $G = (V, E)$ is a pair (v, i), where $v \in V$ is a node and $i \in [\deg(v)]$ is the number of the port. The set of all ports of graph G is denoted by $P(G)$. Now, a *port numbering* for G is a bijective function $p \colon P(G) \to P(G)$ such that there exist i and j for which $p(v, i) = (u, j)$ if and only if $\{v, u\} \in E$. That is, adjacent nodes – and only those – are connected by the port numbering. If $p(v, i) = (u, j)$, then (v, i) is an *output port* of node v that is connected to the *input port* (u, j) of node u. If it holds that $p(p(v, i)) = (v, i)$ for all $(v, i) \in P(G)$, we

say that the port numbering P is *consistent*.

Distributed state machines. Let $\Delta \in \mathbb{N}$ and let I be a set of *local node information*. We define a *distributed state machine* for $(\mathcal{F}(\Delta), I)$ to be a tuple $\mathcal{A} = (S, H, \sigma_0, M, \mu, \sigma)$, where

- S is a set of states,
- $H \subseteq S$ is a set of stopping (or halting) states,
- $\sigma_0 \colon I \to S$ gives the initial state,
- M is a set of messages, with $\emptyset \in M$,
- $\mu \colon S \times [\Delta] \to M$ constructs the outgoing messages, with $\mu(s, i) = \emptyset$ for all $s \in H$ and $i \in [\Delta]$,
- $\sigma \colon S \times M^\Delta \to S$ defines the state transitions, with $\sigma(s, \overline{m}) = s$ for all $s \in H$ and $\overline{m} \in M^\Delta$.

The elements of the set I can be used to encode problem-specific local input labels, as well as model-specific information such as node degrees or unique node identifiers. This will become clearer as we define the concrete models of computation in Sections 2.3.3 and 2.3.4.

Remark 1. We define distributed state machines for a given upper bound Δ on the node degrees. However, it would be straightforward to generalise the definition for arbitrary-degree graphs, at the expense of not necessarily having a finite description for the functions μ and σ. As the general case is not relevant in the present work, and the bounded-degree requirement is actually necessary for the results of Chapter 3, we leave it to the reader to imagine the generalisation.

Executions. Given a graph $G = (V, E) \in \mathcal{F}(\Delta)$, a port numbering p for G, a mapping $f \colon V \to I$ of local node information, and a distributed state machine $\mathcal{A}$ for $(\mathcal{F}(\Delta), I)$, we can define the *execution* of $\mathcal{A}$ on (G, p, f) inductively as follows.

We denote the state of node v in round r by $x_r(v)$, where $x_r \colon V \to S$ encodes the state of the system in round $r \in \mathbb{N}$. The initial states are given by $x_0(v) = \sigma_0(f(v))$ for each $v \in V$. Then, assume that $x_r \colon V \to S$ is defined for some $r \in \mathbb{N}$. Assume that $(v, i) = p(u, j)$ for some $(u, j) \in P(G)$. Let $a_{r+i}(v, i) = \mu(x_r(u), j) \in M$ be the message that node v receives through its port (v, i) from its neighbour u in round $r + 1$. Now, for each $v \in V$, we set

$$\overline{a}_{r+1}(v) = (a_{r+1}(v, 1), a_{r+1}(v, 2), \ldots, a_{r+1}(v, \deg(v)), \emptyset, \emptyset, \ldots, \emptyset) \in M^\Delta,$$

that is, $\overline{a}_{r+1}(v)$ contains all the messages received by node v in round $r + 1$, as well as $\Delta - \deg(v)$ instances of the dummy value $\emptyset$ so that $\overline{a}_{r+1} \colon V \to M^\Delta$. Now, the state of each node v in round $r + 1$ is simply

$$x_{r+1}(v) = \sigma(x_r(v), \overline{a}_{r+1}(v)).$$

Note that by the definition of distributed state machines, once a node v reaches one of the halting states $s \in H$, it stays in the state s indefinitely and keeps sending the dummy message $\emptyset$ – this essentially means that the execution is terminated on the node v.

The execution of $\mathscr{A}$ on (G,p) is defined to be the execution of $\mathscr{A}$ on (G,p,f), where $f(v) = \deg(v)$ for all $v \in V$, that is, nodes are given their degrees and nothing else as local node information.

Complexity measures and outputs. A distributed state machine $\mathscr{A}$ is said to *halt* on (G,p,f) if there exists $t \in \mathbb{N}$ such that $x_t(v) \in H$ holds for all $v \in V$. If $\mathscr{A}$ halts on (G,p,f), we define the *running time* of $\mathscr{A}$ on (G,p,f) to be the smallest $t \in \mathbb{N}$ for which this holds. Otherwise, the running time is defined to be ∞. That is, the running time is the number of communication rounds needed until all nodes have entered one of the halting states.

On the other hand, the *space usage* of $\mathscr{A}$ on (G,p,f), assuming that $\mathscr{A}$ halts on (G,p,f), is defined to be

$$\left\lceil \log_2 \left| \{x_r(v) \in S : r \in [0,t] \text{ and } v \in V\} \right| \right\rceil,$$

where $t \in \mathbb{N}$ is the running time of $\mathscr{A}$ on (G,p,f). That is, the space usage is defined to be the number of bits needed to encode all the states visited by any node during the execution.

If the running time of $\mathscr{A}$ on (G,p,f) is $t \in \mathbb{N}$, the *output* of $\mathscr{A}$ on (G,p,f) is $x_t \colon V \to H$. For each node $v \in V$, the *local output* of v is $\mathscr{A}(v) = x_t(v)$.

2.3.3 The port-numbering model

Given the definitions of Section 2.3.2, we can finally introduce our first model of computation, the *port-numbering* model [2]. In this model, nodes are *anonymous* – that is, they do not have any kind of unique identifiers to distinguish one node from another. Instead, as the name suggests, they can make use of a port numbering to distinguish their communication ports.

Solving graph problems. Some technicalities are needed to handle local inputs. Let $\Delta \in \mathbb{N}$, let $G = (V,E) \in \mathscr{F}(\Delta)$ be a graph, let $i \colon V \cup \vec{E} \to \Sigma$ be an input labelling for G, and let p be a port numbering of G. For each node $u \in V$ and each $j \in [\deg(u)]$, denote by $v_{u,j}$ the neighbour of u for which $p(u,j) = (v_{u,j}, j')$ for some j'. Similarly, denote by $w_{u,j}$ the neighbour of u for which $p(w_{u,j}, j') = (u,j)$ for some j'. That is, $v_{u,1}, v_{u,2}, \ldots, v_{u,\deg(u)}$ is the enumeration of the neighbours of u in the order given by outgoing port numbers, whereas in $w_{u,1}, w_{u,2}, \ldots, w_{u,\deg(u)}$, the order is given by incoming port numbers.

Now, set

$$\overline{e}_{\mathrm{o}}(u) = (i(u,v_{u,1}), i(u,v_{u,2}), \ldots, i(u,v_{u,\deg(u)}), \emptyset, \emptyset, \ldots, \emptyset) \in \Sigma^{\Delta},$$

and similarly,

$$\overline{e}_{\mathrm{i}}(u) = (i(u,w_{u,1}), i(u,w_{u,2}), \ldots, i(u,w_{u,\deg(u)}), \emptyset, \emptyset, \ldots, \emptyset) \in \Sigma^{\Delta}.$$

Note that in case of a consistent port numbering, we have $\overline{e}_{\mathrm{o}} = \overline{e}_{\mathrm{i}}$ – we consider the more general case here to demonstrate that our framework of local inputs can also handle other model variants. Define then a function $f_{i,p} \colon V \to [0,\Delta] \times \Sigma \times \Sigma^{\Delta} \times \Sigma^{\Delta}$ by setting

$$f_{i,p}(v) = (\deg_G(v), i(v), \overline{e}_{\mathrm{o}}, \overline{e}_{\mathrm{i}})$$

for each $v \in V$. That is, $f_{i,p}$ is a local node information mapping that gives the degree as well as the local input label and the input labels of incident edges for each node. Here $I_\Delta = [0,\Delta] \times \Sigma \times \Sigma^{\Delta} \times \Sigma^{\Delta}$.

Finally, let $\Pi_{\Sigma,\Gamma}$ be a graph problem, $T, T' \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ functions, $\mathscr{G}$ a class of graphs and $\mathbf{A} = (\mathscr{A}_1, \mathscr{A}_2, \ldots)$ a sequence such that each $\mathscr{A}_\Delta$ is a distributed state machine for $(\mathscr{F}(\Delta), I_\Delta)$. We define that $\mathbf{A}$ *solves* $\Pi_{\Sigma,\Gamma}$ *in class* $\mathscr{G}$ *in time* T *and space* T' *in the port-numbering model* if the following holds for all $\Delta \in \mathbb{N}$, all graphs $G = (V,E) \in \mathscr{G} \cap \mathscr{F}(\Delta)$, all consistent port numberings p for G and all input labellings $i \colon V \cup \vec{E} \to \Sigma$:

(1) the output of $\mathscr{A}_\Delta$ on $(G, p, f_{i,p})$ is in $\Pi_{\Sigma,\Gamma}(G, i)$,
(2) the running time of $\mathscr{A}_\Delta$ on $(G, p, f_{i,p})$ is at most $T(|V|, \Delta)$,

and furthermore,

$$T'(n,\Delta) = \left\lceil \log_2 \left| \mathscr{S} \right| \right\rceil \text{ for all } n, \Delta \in \mathbb{N},$$

where the set $\mathscr{S}$ consists of all the states $x_r(v) \in S$ that $\mathscr{A}_\Delta$ visits during its execution on $(G, p, f_{i,p})$ when

- G ranges over all graphs with n nodes in $\mathscr{G} \cap \mathscr{F}(\Delta)$,
- p ranges over all consistent port numberings for G,
- i ranges over all input labellings $i \colon V \cup \vec{E} \to \Sigma$.

We also say that $\mathbf{A}$ is a *port-numbering algorithm* for problem $\Pi_{\Sigma,\Gamma}$ in class $\mathscr{G}$.

From now on, we will informally refer to both distributed state machines $\mathscr{A}$ and sequences of distributed state machines $\mathbf{A}$ as *algorithms* – the precise meaning is revealed by the notation.

Remark 2. Note that in the port-numbering model, nodes can find out their own degree in one communication round by simply counting the number of messages they receive. However, later we will consider restricted versions of the model, where this is not true. Hence it makes sense to give the degree of each node as part of the local node information.

Problem complexity. Consider a graph problem $\Pi_{\Sigma,\Gamma}$ and a class $\mathscr{G}$ of graphs. For each $n \in \mathbb{N}$, let $T_n \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function with the following properties:

(1) There exists an algorithm $\mathbf{A}$ that solves $\Pi_{\Sigma,\Gamma}$ in class $\mathscr{G}$ in time (space) T_n in the port-numbering model.

(2) For any algorithm $\mathbf{A}'$ that solves $\Pi_{\Sigma,\Gamma}$ in class $\mathscr{G}$ in time (space) T' in the port-numbering model, we have $T'(n,\Delta) \geq T_n(n,\Delta)$ for all $\Delta \in \mathbb{N}$.

Now, the *time (space) complexity of problem* $\Pi_{\Sigma,\Gamma}$ *on class* $\mathcal{G}$ *in the port-numbering model* is defined to be the function $T\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N}$ where $T(n,\Delta) = T_n(n,\Delta)$ for all $n,\Delta\in\mathbb{N}$. That is, the time complexity of a problem equals the running time of the fastest algorithm solving it – allowing for the fact that different algorithms could be fastest for different values of n and Δ. Somewhat analogously, the space complexity of a problem equals the number of bits needed to encode all the states that the most space-efficient algorithm solving it visits on any input instance of given size and maximum degree.

Often we are only interested in time and space complexity as a function of n in bounded-degree graphs, in which case we usually make use of the simplified notation $T\colon \mathbb{N}\to\mathbb{N}$.

2.3.4 The LOCAL model

In the LOCAL model [28, 31], nodes are equipped with unique identifiers. This makes the model significantly more powerful than the port-numbering model defined in Section 2.3.3, since symmetry breaking between nodes is always possible. The size of the identifiers is limited by some fixed polynomial $q\colon\mathbb{N}\to\mathbb{N}$ in the size of the graph, so that each node v has an identifier $\mathrm{id}(v)\in[0,q(n)-1]$, where $n=|V|$ is the number of nodes in the graph. Such an injective function $\mathrm{id}\colon V\to\mathbb{N}$ is called an *identifier assignment* for a graph $G=(V,E)$.

In addition to identifiers, we provide nodes with knowledge of the number n of nodes in the graph. While it makes sense to also study variants of the LOCAL model where nodes are given only a polynomial upper bound on n or no information at all, our results presented in Chapter 5 are based on this stronger variant.

Given an input labelling $i\colon V\cup\vec{E}\to\Sigma$, a consistent port numbering p and an identifier assignment $\mathrm{id}\colon V\to\mathbb{N}$ for a graph $G=(V,E)\in\mathcal{F}(\Delta)$, define $\overline{e}_{\mathrm{o}} = \overline{e}_{\mathrm{i}} = \overline{e}$ as in the case of the port-numbering model above. Define then a function $f_{G,i,p,\mathrm{id}}\colon V\to\mathbb{N}\times\mathbb{N}\times\Sigma\times\Sigma^{\Delta}$ by setting

$$f_{G,i,p,\mathrm{id}}(v) = (\mathrm{id}(v),|V|,i(v),\overline{e})$$

for each $v\in V$. That is, $f_{G,i,p,\mathrm{id}}$ is a local node information mapping that gives the unique identifier and the number $|V|$ (instead of the degree, as in the case of the port-numbering model) as well as the local input labels for each node.

Now, let $\Pi_{\Sigma,\Gamma}$ be a graph problem, $T\colon\mathbb{N}\times\mathbb{N}\to\mathbb{N}$ a function, $\mathcal{G}$ a class of graphs and $\mathbf{A}=(\mathcal{A}_1,\mathcal{A}_2,\ldots)$ a sequence such that each $\mathcal{A}_\Delta$ is a distributed state machine for $(\mathcal{F}(\Delta),\mathbb{N}\times\mathbb{N}\times\Sigma\times\Sigma^{\Delta})$. We define that $\mathbf{A}$ *solves* $\Pi_{\Sigma,\Gamma}$ *in class* $\mathcal{G}$ *in time* T *and space* T' *in the* LOCAL *model* if the following holds for all $\Delta\in\mathbb{N}$, all graphs $G=(V,E)\in\mathcal{G}\cap\mathcal{F}(\Delta)$, all consistent port numberings p for G, all identifier assignments $\mathrm{id}\colon V\to\mathbb{N}$ and all input labellings $i\colon V\cup\vec{E}\to\Sigma$:

(1) the output of $\mathcal{A}_\Delta$ on $(G,p,f_{G,i,p,\mathrm{id}})$ is in $\Pi_{\Sigma,\Gamma}(G,i)$,
(2) the running time of $\mathcal{A}_\Delta$ on $(G,p,f_{G,i,p,\mathrm{id}})$ is at most $T(|V|,\Delta)$,

and furthermore,

$$T'(n,\Delta) = \left\lceil \log_2 \left| \mathscr{S} \right| \right\rceil \text{ for all } n, \Delta \in \mathbb{N},$$

where the set $\mathscr{S}$ consists of all the states $x_r(v) \in S$ that $\mathscr{A}_\Delta$ visits during its execution on $(G, p, f_{G,i,p,\mathrm{id}})$ when

- G ranges over all graphs with n nodes in $\mathscr{G} \cap \mathscr{F}(\Delta)$,
- p ranges over all consistent port numberings for G,
- i ranges over all input labellings $i \colon V \cup \vec{E} \to \Sigma$,
- id ranges over all identifier assignments $\mathrm{id} \colon V \to \mathbb{N}$.

We also say that $\mathbf{A}$ is a LOCAL *algorithm* for problem $\Pi_{\Sigma,\Gamma}$ in class $\mathscr{G}$.

Remark 3. In our definition of the LOCAL model, we assume that a port numbering is given – to make the definitions of the models of computation more uniform. However, this is not strictly necessary.

In one communication round, each node v can gather the unique identifiers of its neighbours. Then, virtual port numbers can be constructed by forming a bijective mapping between the identifiers of the neighbours and the set $\{1, 2, \ldots, \deg(v)\}$. Now it is possible to simulate port numbers by broadcasting a message that contains the identifier $\mathrm{id}(v)$ of the sending node v as well as the identifier $\mathrm{id}(u)$ of each intended receiver u together with the message for neighbour u.

To allow input labels that nodes associate to incident edges, the definition would need to make use of unique identifiers to connect the labels to corresponding edges.

Problem complexity. The time and space complexity of graph problems are defined for the LOCAL model in a manner completely analogous to the port-numbering model above.

Locally checkable labelling (LCL) problems. The *locally checkable labelling* (LCL) problems [30] are a very important subclass of graph problems when it comes to the LOCAL model. Intuitively, a problem is an LCL problem if its solutions can be verified by a constant-time LOCAL algorithm.

More formally, a graph problem $\Pi_{\Sigma,\Gamma}$ is an LCL problem if the following holds:

(1) The sets Σ and Γ of input and output labels are finite.

(2) There exists a LOCAL algorithm $\mathbf{A}$ with a running time independent of the size of the input graph, such that given a graph $G = (V, E) \in \mathscr{F}(\Delta)$, an input labelling $i \colon V \cup \vec{E} \to \Sigma$, and a candidate solution $i' \colon V \to \Gamma$ as another input labelling, we have

$$\mathscr{A}_\Delta(v) = 1 \text{ for all } v \in V \quad \text{ if and only if } \quad i' \in \Pi_{\Sigma,\Gamma}(G, i).$$

LCL problems can also be described by simply enumerating all the *good* neighbourhoods – that is, labelled neighbourhoods that the algorithm $\mathscr{A}_\Delta$ above would

accept. Since the label sets are finite and we are working with bounded-degree graphs, there is a finite number of such constant-size neighbourhoods. Yet another way to define LCL problems is by *non-deterministic* algorithms: LCL problems are those that can be solved by constant-time LOCAL algorithms that have the additional ability to make non-deterministic guesses.

2.4 Modal logic

In this section, we give an introduction to *modal logic* [4, 5] – a non-classical logic that we will use to characterise several models of distributed computing in Chapter 3. While there exists several variants and extensions of modal logic, we will concentrate on the *basic modal logic* for now. Other variants that correspond to different models of computation will be introduced in Chapter 3. We will assume that the reader is familiar with classical propositional logic.

2.4.1 Syntax

The basic modal logic extends propositional logic by adding two new operators: $\Diamond$ ("*diamond*") and $\Box$ ("*box*"). Given a (finite or countably infinite) set Φ of proposition symbols, the *alphabet* (or *vocabulary*) of the *basic modal logic* consists of the following primitive symbols: proposition symbols $p \in \Phi$, the *negation* symbol $\neg$, the *disjunction* symbol $\vee$, parentheses) and (as well as the diamond $\Diamond$. The set of *(well-formed) formulas* of basic modal logic is now defined inductively as follows:

(1) Each proposition symbol $p \in \Phi$ is a formula.
(2) If ϕ is a formula, then its negation $\neg\phi$ is a formula.
(3) If ϕ and ψ are formulas, then their disjunction $(\phi \vee \psi)$ is a formula.
(4) If ϕ is a formula, then $\Diamond\phi$ is a formula.

Conjunction, implication and equivalence can be defined as abbreviations by using the primitive symbols in the usual way:

$$(\phi \wedge \psi) = \neg(\neg\phi \vee \neg\psi),$$

$$(\phi \to \psi) = (\neg\phi \vee \psi),$$

$$(\phi \leftrightarrow \psi) = ((\phi \to \psi) \wedge (\psi \to \phi)).$$

Finally, the other modal operator $\Box$ is defined as the *dual* of $\Diamond$:

$$\Box\phi = \neg\Diamond\neg\phi.$$

Parentheses are usually written out only when they are necessary to make the formula unambiguous.

2.4.2 Semantics

To characterise the truth behaviour of modal formulas, we will next introduce *relational semantics* – also known as *Kripke semantics*. Relational semantics is related to the philosophical concept of *possible worlds*, but provides a mathematically rigorous way to define truth in modal logic. It also enables the use of modal logic as a tool for studying relational structures.

A *frame* for the basic modal logic is a pair $F = (W, R)$, where W is a non-empty set of *points* and $R \subseteq W \times W$ is an arbitrary binary relation on W. The set W is called the *domain* of F and the relation R is called the *accessibility relation* of F. The elements of W are sometimes called *states*, *worlds*, *times* or *situations*, depending on the context. Note that a frame is essentially a directed graph. If $(w, w') \in R$, point w' is said to be *accessible* from point w. We denote the set of points accessible from w by $R(w) = \{w' \in W : (w, w') \in R\}$.

A *model* for the basic modal logic is a pair $M = (F, \tau)$, where F is a frame for the basic modal logic and τ is a function $\Phi \to \mathscr{P}(W)$. The function τ is called the *valuation* of M and it assigns a subset $\tau(p) \subseteq W$ of points to each proposition symbol $p \in \Phi$. If $F = (W, R)$, we can simply write $M = (W, R, \tau)$.

Now we are ready to define the notion of a formula being *satisfied* or *true*. In modal logic, satisfaction is defined in a particular point of a model. Given a model $M = (W, R, \tau)$ and its point $w \in W$, we define the truth of a formula ϕ in M at point w inductively as follows:

$$\begin{aligned}
\phi = p: &\quad M, w \vDash \phi \text{ iff } w \in \tau(p), \\
\phi = \neg\psi: &\quad M, w \vDash \phi \text{ iff } M, w \nvDash \psi, \\
\phi = \psi_1 \vee \psi_2: &\quad M, w \vDash \phi \text{ iff } M, w \vDash \psi_1 \text{ or } M, w \vDash \psi_2, \\
\phi = \Diamond\psi: &\quad M, w \vDash \phi \text{ iff } M, v \vDash \psi \text{ for some } v \in W \text{ with } (w, v) \in R.
\end{aligned}$$

If $M, w \vDash \phi$, we say that formula ϕ is *satisfied* or *true* in model M at point w, and conversely, if $M, w \nvDash \phi$, we say that ϕ is *false* in M at point w.

Observe that the dual of the diamond operator, box, gets the following semantics from the previous definition:

$$M, w \vDash \Box\psi \text{ iff } M, v \vDash \psi \text{ for all } v \in W \text{ with } (w, v) \in R.$$

Additionally, formulas of the form $\psi_1 \wedge \psi_2$, $\psi_1 \to \psi_2$ and $\psi_1 \leftrightarrow \psi_2$ get the usual semantics that are familiar from propositional logic.

Given a model $M = (W, R, \tau)$ and a formula ϕ, we say that ϕ *defines* the subset

$$\|\phi\|^M = \{w \in W : M, w \vDash \phi\}$$

of the domain of M.

2.4.3 Bisimulation

One of the most important concepts in modal logic is *bisimulation* [4]. A bisimulation is a relation between two structures such that related points satisfy the same proposition symbols and have access to points which are similarly related. Bisimilarity characterises definability in modal logic: if two structures are *bisimilar*, they cannot be distinguished by any modal formula.

Definition 4. Let $M = (W, R, \tau)$ and $M' = (W', R', \tau')$ be models for the basic modal logic. A non-empty binary relation $B \subseteq W \times W'$ is a *bisimulation*, if the following conditions hold for all $(w, w') \in B$:

(1) For all proposition symbols $p \in \Phi$, we have $w \in \tau(p)$ iff $w' \in \tau'(p)$.
(2) If $(w, v) \in R$, then there exists $v' \in W'$ with $(w', v') \in R'$ such that $(v, v') \in B$.
(3) If $(w', v') \in R'$, then there exists $v \in W$ with $(w, v) \in R$ such that $(v, v') \in B$.

If $(w, w') \in B$ for some bisimulation B, we say that points w and w' are *bisimilar* and write $(M, w) \leftrightarrows (M', w')$.

The next lemma states that bisimilar points are *modally equivalent*.

Lemma 5. *Let $M = (W, R, \tau)$ and $M' = (W', R', \tau')$ be models for the basic modal logic, and let $w \in W$ and $w' \in W'$. If $(M, w) \leftrightarrows (M', w')$, then for all formulas ϕ of the basic modal logic we have*

$$M, w \vDash \phi \quad \textit{iff} \quad M', w' \vDash \phi.$$

Proof idea. Straightforward induction on the structure of the formula. □

The converse does not hold in general: modal equivalence does not imply bisimilarity. However, the converse does hold for all finite models [4].

Next, we will consider a restricted version of bisimilarity that we call *r-bisimilarity*, where r is a natural number. Here the idea is that points are required to be bisimilar only up to a certain depth in the model.

Definition 6. Let $M = (W, R, \tau)$ and $M' = (W', R', \tau')$ be models for the basic modal logic. We define r-bisimilarity recursively as follows. As a base case, we say that points $w \in W$ and $w' \in W'$ are 0-*bisimilar* if for all proposition symbols $p \in \Phi$, we have $w \in \tau(p)$ iff $w' \in \tau'(p)$. For $r \in \mathbb{N}_+$, we say that $w \in W$ and $w' \in W'$ are r-*bisimilar* if the following conditions hold:

(1) Points w and w' are 0-bisimilar.

(2) If $(w, v) \in R$, then there exists $v' \in W'$ with $(w', v') \in R'$ such that v and v' are $(r-1)$-bisimilar.

(3) If $(w', v') \in R'$, then there exists $v \in W$ with $(w, v) \in R$ such that v and v' are $(r-1)$-bisimilar.

If $w \in W$ and $w' \in W'$ are r-*bisimilar*, we write $(M, w) \leftrightarrows_r (M', w')$.

The corresponding notion for formulas is modal depth. The *modal depth* $\mathrm{md}(\phi)$ of a formula ϕ is defined recursively as follows:

$$\begin{aligned}
\phi = p: &\quad \mathrm{md}(\phi) = 0,\\
\phi = \neg\psi: &\quad \mathrm{md}(\phi) = \mathrm{md}(\psi),\\
\phi = \psi_1 \vee \psi_2: &\quad \mathrm{md}(\phi) = \max\{\mathrm{md}(\psi_1), \mathrm{md}(\psi_2)\},\\
\phi = \Diamond\psi: &\quad \mathrm{md}(\phi) = \mathrm{md}(\psi) + 1.
\end{aligned}$$

That is, $\mathrm{md}(\phi)$ indicates the largest number of nested modal operators in ϕ. Now, the following result holds:

Lemma 7. *Let $M = (W, R, \tau)$ and $M' = (W', R', \tau')$ be models for the basic modal logic, and let $w \in W$ and $w' \in W'$. If $(M, w) \leftrightarrows_r (M', w')$, then for all formulas ϕ of the basic modal logic with $\mathrm{md}(\phi) \leq r$ we have*

$$M, w \vDash \phi \quad \textit{iff} \quad M', w' \vDash \phi.$$

Proof idea. Straightforward induction on the structure of the formula. □

3. Distributed Computing and Modal Logic

In this chapter, we will study a natural connection between several models of distributed computation and variants of modal logic. The models considered are weaker variants of the port-numbering model defined in Section 2.3.3. First, we will define the models of computation in Section 3.1. Then, we will outline the connection to modal logic in Section 3.2. Finally, we will give results on the relationships between the models in Section 3.3.

3.1 A hierarchy of weak models

In this section, we will give several different ways to restrict the communication capabilities of the port-numbering model. This will result in a collection of seven model variants.

To study the relationships between the models, it is natural to equate the models of computation with the classes of graph problems that are solvable in each model. To this end, we denote by VV_c the class of all graph problems that are solvable in the port-numbering model. The intuition behind the notation is that on each round, nodes basically send a *vector* of messages and receive a *vector* of messages – and the order of messages in those vectors is *consistent*.

The weaker model variants and corresponding problem classes that we study are as follows:

- VV: Input and output ports are numbered, but not necessarily consistently.
- MV: Output ports are numbered; nodes receive a *multiset* of messages.
- SV: Output ports are numbered; nodes receive a *set* of messages.
- VB: Input ports are numbered; nodes *broadcast* one message to all neighbours.
- MB: Combination of the restrictions of MV and VB.
- SB: Combination of the restrictions of SV and VB.

By slight abuse of notation, we will use the above classes to refer to the corresponding models of computation from now on.

3.1.1 Definitions of the models

Let us now formally define the six restricted versions of the port-numbering model. Recall the definitions of a distributed state machine and the port-numbering model from Sections 2.3.2 and 2.3.3.

Given a vector $\overline{a} = (a_1, a_2, \ldots, a_\Delta) \in M^\Delta$, we write

$$\text{set}(\overline{a}) = \{a_1, a_2, \ldots, a_\Delta\},$$

$$\text{multiset}(\overline{a}) = \{(m, n) : m \in M, n = |\{i \in [\Delta] : m = a_i\}|\}.$$

The intuition is that $\text{set}(\overline{a})$ discards the ordering and multiplicities of the elements of vector $\overline{a}$, while $\text{multiset}(\overline{a})$ discards only the ordering.

Classes of state machines. Let us denote by $\mathcal{VV}$ the class of all distributed state machines as defined in Section 2.3.2 – in essence, on each round a state machine sends a *vector* of messages and receives a *vector* of messages. We will now define several subclasses of $\mathcal{VV}$ as follows.

First, class $\mathcal{MV} \subseteq \mathcal{VV}$ consists of those state machines that consider only the *multiset* of messages received on each round:

$$\mathcal{MV} = \{\mathcal{A} \in \mathcal{VV} : \text{multiset}(\overline{a}) = \text{multiset}(\overline{b}) \Rightarrow \sigma(s, \overline{a}) = \sigma(s, \overline{b}) \text{ for all } s \in S\}.$$

A natural further restriction, $\mathcal{SV} \subseteq \mathcal{MV}$, ignores also the multiplicities of identical incoming messages, that is, the messages are received in a *set*:

$$\mathcal{SV} = \{\mathcal{A} \in \mathcal{VV} : \text{set}(\overline{a}) = \text{set}(\overline{b}) \Rightarrow \sigma(s, \overline{a}) = \sigma(s, \overline{b}) \text{ for all } s \in S\}.$$

In addition to restricting the amount of information nodes get regarding received messages, we can also restrict the way they are able to send messages. Class $\mathcal{VB} \subseteq \mathcal{VV}$ consists of those state machines that on each round *broadcast* the same message to all the neighbours of a node:

$$\mathcal{VB} = \{\mathcal{A} \in \mathcal{VV} : \mu(s, i) = \mu(s, j) \text{ for all } i, j \in [\Delta] \text{ and } s \in S\}.$$

Finally, we can combine the above restriction to establish two additional classes. In class $\mathcal{MB}$, messages are received in a *multiset* and in class $\mathcal{SB}$, messages are received in a *set*, while in both classes, state machines send messages by *broadcasting*:

$$\mathcal{MB} = \mathcal{MV} \cap \mathcal{VB} \quad \text{and} \quad \mathcal{SB} = \mathcal{SV} \cap \mathcal{VB}.$$

Since we work with bounded-degree graphs, our algorithms are actually *infinite sequences* of state machines – one machine for each possible maximum degree of the communication graph. Hence, we define a class of sequences of state

machines for each of the aforementioned classes as follows:

$$\mathbf{VV} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{VV} \text{ for all } \Delta\},$$

$$\mathbf{MV} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{MV} \text{ for all } \Delta\},$$

$$\mathbf{SV} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{SV} \text{ for all } \Delta\},$$

$$\mathbf{VB} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{VB} \text{ for all } \Delta\},$$

$$\mathbf{MB} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{MB} \text{ for all } \Delta\},$$

$$\mathbf{SB} = \{(\mathcal{A}_1, \mathcal{A}_2, \ldots) : \mathcal{A}_\Delta \in \mathcal{SB} \text{ for all } \Delta\}.$$

The definition of a sequence of distributed state machines solving a graph problem is the same as given in Section 2.3.3 – in the case of **VV**, we just use arbitrary port numberings instead of consistent ones. None of the other classes make use of both outgoing and incoming port numbers, and hence the issue of consistency can be ignored.

Both distributed state machines $\mathcal{A} \in \mathcal{VV}$ and infinite sequences of distributed state machines $\mathbf{A} \in \mathbf{VV}$ can be informally referred to as *algorithms*. The precise meaning can be inferred from the notation.

Classes of graph problems. Next, we define a collection of classes of graph problems – think of them as complexity classes – based on the different restrictions of the port-numbering model. We denote the class of all graph problems by $\boldsymbol{\Pi}$. The seven classes studied in our work are as follows:

$$\mathsf{VV_c} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{VV} \text{ solves } \Pi \text{ assuming consistency}\},$$

$$\mathsf{VV} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{VV} \text{ solves } \Pi \text{ without consistency}\},$$

$$\mathsf{MV} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{MV} \text{ solves } \Pi\},$$

$$\mathsf{SV} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{SV} \text{ solves } \Pi\},$$

$$\mathsf{VB} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{VB} \text{ solves } \Pi\},$$

$$\mathsf{MB} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{MB} \text{ solves } \Pi\},$$

$$\mathsf{SB} = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{SB} \text{ solves } \Pi\}.$$

For each of the above classes, we also define a constant-time variant:

$$\mathsf{VV_c}(1) = \left\{ \Pi \in \boldsymbol{\Pi} : \begin{array}{l} \text{some algorithm } \mathbf{A} \in \mathbf{VV} \text{ solves } \Pi \text{ in constant time} \\ \text{assuming consistency} \end{array} \right\},$$

$$\mathsf{VV}(1) = \left\{ \Pi \in \boldsymbol{\Pi} : \begin{array}{l} \text{some algorithm } \mathbf{A} \in \mathbf{VV} \text{ solves } \Pi \text{ in constant time} \\ \text{without consistency} \end{array} \right\},$$

$$\mathsf{MV}(1) = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{MV} \text{ solves } \Pi \text{ in constant time}\},$$

$$\mathsf{SV}(1) = \{\Pi \in \boldsymbol{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{SV} \text{ solves } \Pi \text{ in constant time}\},$$

$$\mathsf{VB}(1) = \{\Pi \in \mathbf{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{VB} \text{ solves } \Pi \text{ in constant time}\},$$

$$\mathsf{MB}(1) = \{\Pi \in \mathbf{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{MB} \text{ solves } \Pi \text{ in constant time}\},$$

$$\mathsf{SB}(1) = \{\Pi \in \mathbf{\Pi} : \text{some algorithm } \mathbf{A} \in \mathbf{SB} \text{ solves } \Pi \text{ in constant time}\}.$$

3.2 Characterisations by logics

In this section, we present a connection between modal logic and the weak models of computation considered in this work. For each of the seven models, we establish a corresponding variant of modal logic that captures the model.

3.2.1 Variants of modal logic

Recall the definition of the basic modal logic ML in Section 2.4. The other variants used in this work are extensions of ML. We will now introduce them by explaining the modifications that need to be made to the definitions given for the basic modal logic.

Graded modal logic GML**.** The *graded modal logic* GML extends basic modal logic with the ability to count. This is achieved by adding graded modal operators $\Diamond_{\geq k}$, $k \in \mathbb{N}$, to the alphabet. The corresponding rule for creating formulas is as follows:

If ϕ is a formula, then $\Diamond_{\geq k}\phi$, where $k \in \mathbb{N}$, is a formula.

Frames and models for GML are identical to those for the basic modal logic. The definition of satisfaction of a formula ϕ in model $M = (W, R, \tau)$ at point $w \in W$ is amended by rule

$$\phi = \Diamond_{\geq k}\psi\text{:} \quad M, w \vDash \phi \text{ iff } |\{v \in W : (w,v) \in R \text{ and } M, v \vDash \psi\}| \geq k$$

for each $k \in \mathbb{N}$. That is, $\Diamond_{\geq k}\psi$ is true at point w if and only if the number of points that are accessible from w and that satisfy ψ is at least k.

Multimodal logic MML**.** The *multimodal logic* MML extends basic modal logic by allowing multiple primitive modal operators $\langle\alpha\rangle$, where $\alpha \in I$ for some index set I, instead of just a single modality $\Diamond$. That is, we have the following additional rule for creating formulas:

If ϕ is a formula, then $\langle\alpha\rangle\phi$, where $\alpha \in I$, is a formula.

A *frame* for the multimodal logic is of the form $F = (W, (R_\alpha)_{\alpha \in I})$, with $W \neq \emptyset$ and $R_\alpha \subseteq W \times W$ for each $\alpha \in I$. That is, there is a separate accessibility relation for each modality $\langle\alpha\rangle$. As previously, we get a *model* $M = (F, \tau)$ for the multimodal logic by adding a valuation $\tau : \Phi \to \mathscr{P}(W)$ to a frame.

The satisfaction of the above kind of formulas in model $M = (W, (R_\alpha)_{\alpha \in I}, \tau)$ at point $w \in W$ is naturally defined as follows:

$$\phi = \langle \alpha \rangle \psi: \quad M, w \vDash \phi \text{ iff } M, v \vDash \psi \text{ for some } v \in W \text{ with } (w, v) \in R_\alpha$$

for each $\alpha \in I$.

Graded multimodal logic GMML**.** Lastly, the *graded multimodal logic* GMML combines the features of GML and MML. To achieve this, we extend the basic modal logic by the following rule:

If ϕ is a formula, then $\langle \alpha \rangle_{\geq k} \phi$, where $\alpha \in I$ and $k \in \mathbb{N}$, is a formula,

where I is some index set. Frames and models for GMML are identical to those for the multimodal logic. The new rule in the definition of satisfaction of a formula ϕ in model $M = (W, (R_\alpha)_{\alpha \in I}, \tau)$ at point $w \in W$ is now

$$\phi = \langle \alpha \rangle_{\geq k} \psi: \quad M, w \vDash \phi \text{ iff } |\{v \in W : (w, v) \in R_\alpha \text{ and } M, v \vDash \psi\}| \geq k$$

for each $\alpha \in I$ and $k \in \mathbb{N}$.

Given an index set I and a set Φ of proposition symbols, the pair (I, Φ) is called a *signature*. A formula ϕ of a logic $\mathscr{L} \in \{\mathrm{ML}, \mathrm{GML}, \mathrm{MML}, \mathrm{GMML}\}$ is said to be *in the signature* (I, Φ) if it is constructed from modal operators given by the index set I and proposition symbols taken from Φ. Note that for ML and GML, set I is a singleton.

Bisimulation. For each of the variants of modal logic, there is a corresponding notion of bisimulation. For GML, the notion is called *graded bisimulation* and defined as follows.

Definition 8. Let $M = (W, R, \tau)$ and $M' = (W', R', \tau')$ be models for the graded modal logic. A non-empty binary relation $B \subseteq W \times W'$ is a *graded bisimulation*, if the following conditions hold for all $(w, w') \in B$:

(1) For all proposition symbols $p \in \Phi$, we have $w \in \tau(p)$ iff $w' \in \tau'(p)$.

(2) If $X \subseteq R(w)$, then there exists $X' \subseteq R'(w')$ such that $|X'| = |X|$ and for each $v' \in X'$ there is $v \in X$ with $(v, v') \in B$.

(3) If $X' \subseteq R'(w')$, then there exists $X \subseteq R(w)$ such that $|X| = |X'|$ and for each $v \in X$ there is $v' \in X'$ with $(v, v') \in B$.

If $(w, w') \in B$ for some graded bisimulation B, we say that points w and w' are *g-bisimilar* and write $(M, w) \leftrightarrows_{\mathrm{g}} (M', w')$.

The notions of bisimulation for MML and graded bisimulation for GMML are the obvious generalisations of the above – we have an accessibility relation R_α for each $\alpha \in I$.

For each of GML, MML and GMML, the *modal depth* of a formula is defined similarly as for the basic modal logic in Section 2.4.3: each generalised modality $\langle \alpha \rangle_{\geq k}$ affects the modal depth in the same way as the basic modality $\Diamond$.

3.2.2 Correspondence between formulas and algorithms

We will now proceed to characterise the correspondence between the weak models of distributed computing defined in Section 3.1 and the logics ML, GML, MML and GMML.

The idea behind the correspondence is as follows. Given a graph $G = (V, E)$, a port numbering p for G and an input labelling i for G, we can construct a relational model $M(G, p, i) = (W, (R_\alpha)_{\alpha \in I}, \tau)$ that encodes the same information. Here,

- points in W correspond to nodes in V,
- accessibility relations R_α correspond to edge set E and port numbering p,
- valuation τ corresponds to node degrees of G as well as to input labelling i.

Let $\mathcal{A}$ be a constant-time distributed algorithm with binary output. Now, the execution of $\mathcal{A}$ on (G, p, i) can be simulated by a modal logic formula ϕ on the model $M(G, p, i)$ – and conversely, the evaluation of ϕ on $M(G, p, i)$ can be done by algorithm $\mathcal{A}$. We want ϕ to be true at point w if and only if $\mathcal{A}$ outputs 1 on the corresponding node. The essential idea is that diamond formulas $\langle \alpha \rangle \psi$ can be interpreted as communication between nodes:

$$M, w \models \langle \alpha \rangle \psi$$

holds if and only if w receives the message "ψ is true" from some w' such that $(w, w') \in R_\alpha$. This implies that in our correspondence, the modal depth of formula ϕ equals the running time of algorithm $\mathcal{A}$.

Without loss of generality, we assume that algorithms produce binary output, that is, the set of halting states is $H = \{0, 1\}$. More complex output labels can be handled by defining one formula for each output bit.

In what follows, we will define the correspondence in more detail and give a high-level outline of the proof.

From port-numbered graphs to relational models. Let us now define the relational model $M(G, p, i)$ based on a graph G, port numbering p for G and input labelling i for G in more detail. We need in fact four different versions of the model $M(G, p, i)$ to accommodate the fact that algorithms in the lower classes cannot use all the information that a port numbering encodes. Fix the upper bound Δ for the maximum degree. Given a graph $G = (V, E) \in \mathcal{F}(\Delta)$ and a port numbering p for G, we define the following accessibility relations:

$$R_{(j,j')} = \{(u, v) \in V \times V : p((v, j')) = (u, j)\} \quad \text{for each } (j, j') \in [\Delta] \times [\Delta].$$

Note that these relations encode all the information provided by p – from the frame $(V, (R_{(j,j')})_{(j,j') \in [\Delta] \times [\Delta]})$ we could reconstruct graph G and port numbering p. Next, we define restricted accessibility relations that ignore the information

provided by either outgoing port numbers, incoming port numbers or both:

$$R_{(j,*)} = \bigcup_{j' \in [\Delta]} R_{(j,j')} \qquad \text{for each } j \in [\Delta],$$

$$R_{(*,j')} = \bigcup_{j \in [\Delta]} R_{(j,j')} \qquad \text{for each } j' \in [\Delta],$$

$$R_{(*,*)} = \bigcup_{(j,j') \in [\Delta] \times [\Delta]} R_{(j,j')}.$$

While the accessibility relations encode the port numbers, we use a valuation to encode local node information – that is, node degrees and input labels. Our set of proposition symbols is

$$\Phi_{\Delta,\Sigma} = \{p_{d,l} : (d,l) \in [\Delta] \times \Sigma\}.$$

The valuation $\tau \colon \Phi_{\Delta,\Sigma} \to \mathscr{P}(V)$ is defined by

$$\tau(p_{d,l}) = \{v \in V : \deg(v) = d \text{ and } i(v) = l\}.$$

Finally, we are ready to define the four variants of the relational model corresponding to graph $G = (V,E)$, port numbering p for G and input labelling i for G:

$$M_{+,+}(G,p,i) = (V, (R_\alpha)_{\alpha \in I^\Delta_{+,+}}, \tau), \quad \text{where} \quad I^\Delta_{+,+} = [\Delta] \times [\Delta],$$

$$M_{-,+}(G,p,i) = (V, (R_\alpha)_{\alpha \in I^\Delta_{-,+}}, \tau), \quad \text{where} \quad I^\Delta_{-,+} = \{*\} \times [\Delta],$$

$$M_{+,-}(G,p,i) = (V, (R_\alpha)_{\alpha \in I^\Delta_{+,-}}, \tau), \quad \text{where} \quad I^\Delta_{+,-} = [\Delta] \times \{*\},$$

$$M_{-,-}(G,p,i) = (V, (R_\alpha)_{\alpha \in I^\Delta_{-,-}}, \tau), \quad \text{where} \quad I^\Delta_{-,-} = \{(*,*)\}.$$

For $a,b \in \{-,+\}$, we write $\mathscr{M}_{a,b}$ to denote the class of all relational models of the form $M_{a,b}(G,p,i)$, and additionally, we write $\mathscr{M}^{\mathsf{c}}_{+,+}$ to denote the class of relational models $M_{+,+}(G,p,i) \in \mathscr{M}_{+,+}$ where port numbering p is consistent.

Solving graph problems by formulas. Since we use distributed algorithms to solve graph problems, and we want to establish a correspondence between algorithms and formulas, we need to define the concept of formulas that solve graph problems. Recall that we assume binary output labelling: we set $\Gamma = \{0,1\}$ – in essence, each solution $S \in \Pi(G,i)$ defines a subset of the node set V of G.

Let $a,b \in \{-,+\}$. Consider a sequence $\Psi = \{\psi_1, \psi_2, \ldots\}$ of modal formulas, such that each ψ_Δ is in the signature $(I^\Delta_{a,b}, \Phi_{\Delta,\Sigma})$. We say that Ψ *defines a solution* for a graph problem Π on the class $\mathscr{M}_{a,b}$ of models if the following holds:

- For each $\Delta \in \mathbb{N}$, each $G = (V,E) \in \mathscr{F}(\Delta)$, any port numbering p for G and any input labelling i for G, the function $S \colon V \to \Sigma$ defined by setting $S(v) = 1$ if $M_{a,b}(G,p,i), v \vDash \psi_\Delta$, and $S(v) = 0$ otherwise, is in the set $\Pi(G,i)$ of valid solutions.

Additionally, we say that Ψ *defines a solution* for Π on the class $\mathcal{M}^{\mathsf{c}}_{+,+}$, if the above condition holds for $a = b = +$ and all *consistent* port numberings p for G.

Given a sequence $\Psi = \{\psi_1, \psi_2, \ldots\}$ of modal formulas, such that each ψ_Δ is in the signature $(I^\Delta_{a,b}, \Phi_{\Delta,\Sigma})$, we can also construct a canonical graph problem Π_Ψ for which Ψ defines a solution. For each graph $G \in \mathcal{F}(\Delta)$ and each input labelling i for G, the set $\Pi_\Psi(G,i)$ consists of all functions $S\colon V \to \Sigma$ such that for some (consistent) port numbering p for G, we have $S(v) = 1$ if $M_{a,b}(G,p,i), v \vDash \psi_\Delta$ and $S(v) = 0$ otherwise.

Capturing problem classes by logics. Now we have all the ingredients that we need to go into the correspondence proper between logics and models of computation. Let $\mathcal{L}$ be one of the modal logics ML, GML, MML and GMML. Let $a, b \in \{-, +\}$ and let C be a class of graph problems.

On one hand, we say that $\mathcal{L}$ *is contained in* C *on* $\mathcal{M}_{a,b}$ and write $\mathcal{L} \le C$ on $\mathcal{M}_{a,b}$, if for each sequence $\Psi = (\psi_1, \psi_2, \ldots)$ where ψ_Δ is in the signature $(I^\Delta_{a,b}, \Phi_{\Delta,\Sigma})$ for each Δ, we have $\Pi_\Psi \in C$. On the other hand, we say that $\mathcal{L}$ *simulates* C *on* $\mathcal{M}_{a,b}$ and write $C \le \mathcal{L}$ on $\mathcal{M}_{a,b}$, if for every graph problem $\Pi \in C$ there exists a sequence $\Psi = (\psi_1, \psi_2, \ldots)$ where ψ_Δ is in the signature $(I^\Delta_{a,b}, \Phi_{\Delta,\Sigma})$ for each Δ, such that Ψ defines a solution for Π on $\mathcal{M}_{a,b}$.

If both $\mathcal{L} \le C$ and $C \le \mathcal{L}$ on $\mathcal{M}_{a,b}$ hold, we say that $\mathcal{L}$ *captures* C *on* $\mathcal{M}_{a,b}$. Finally, the notions of $\mathcal{L}$ being contained in C on $\mathcal{M}^{\mathsf{c}}_{+,+}$, $\mathcal{L}$ simulating C on $\mathcal{M}^{\mathsf{c}}_{+,+}$ and $\mathcal{L}$ capturing C on $\mathcal{M}^{\mathsf{c}}_{+,+}$ are obtained in a natural way by considering only consistent port numberings.

Now, we state one of the two main results of Publication I.

Theorem 9. (a) MML *captures* $\mathsf{VV_c}(1)$ *on* $\mathcal{M}^{\mathsf{c}}_{+,+}$.
(b) MML *captures* $\mathsf{VV}(1)$ *on* $\mathcal{M}_{+,+}$.
(c) GMML *captures* $\mathsf{MV}(1)$ *on* $\mathcal{M}_{-,+}$.
(d) MML *captures* $\mathsf{SV}(1)$ *on* $\mathcal{M}_{-,+}$.
(e) MML *captures* $\mathsf{VB}(1)$ *on* $\mathcal{M}_{+,-}$.
(f) GML *captures* $\mathsf{MB}(1)$ *on* $\mathcal{M}_{-,-}$.
(g) ML *captures* $\mathsf{SB}(1)$ *on* $\mathcal{M}_{-,-}$.

Overview of the proof. We will now outline the high-level idea behind the proof of Theorem 9. While there are differences in technical details, the general structure of the proof is similar for the cases (a)–(g).

First, to show that a logic $\mathcal{L}$ is contained in class C on $\mathcal{M}_{a,b}$, we need to emulate the recursive evaluation of each formula of $\mathcal{L}$ by an algorithm $\mathcal{A}$ in the corresponding class. To that end, let $\Psi = (\psi_1, \psi_2, \ldots)$ be a sequence of formulas. For each Δ, we simulate the evaluation of formula ψ_Δ by algorithm $\mathcal{A}_\Delta$ on a relational model $M_{a,b}(G,p,i)$ roughly as follows.

Denote the set of all subformulas of ψ_Δ by $\hat{\psi}_\Delta$. Now, in algorithm $\mathcal{A}_\Delta$, we use the state $x_t(v)$ of each node v to keep track of the truth value of each formula $\phi \in \hat{\psi}_\Delta$ with modal depth $\mathrm{md}(\phi)$ at most t. The state set S of $\mathcal{A}_\Delta$ is

$$\left\{ f\colon \hat{\psi}_\Delta \to \{0,1,U\} \right\} \cup \{0,1\}.$$

The idea is that in round t, the state of node v is $x_t(v) = f$, where for each subformula ϕ with $\mathrm{md}(\phi) \leq t$, the value $f(\phi) \in \{0,1\}$ encodes the truth value of ϕ in model $M_{a,b}(G,p,i)$ at point v, and for each subformula ϕ with $\mathrm{md}(\phi) > t$, we have $f(\phi) = U$ indicating that the truth value is *undefined*. The states $\{0,1\} \subseteq S$ are halting states.

The message set M of $\mathscr{A}_\Delta$ consists of functions h that, given a formula ϕ with $\langle\alpha\rangle\phi \in \hat{\psi}_\Delta$ for some suitable α, map formula ϕ to the set $\{0,1,U\}$. The intuition behind this is that to obtain a truth value for a subformula of the form $\langle\alpha\rangle\phi$, node v needs to learn the truth value of ϕ in its neighbours u – and the truth value of ϕ in encoded in the function h sent by u.

The state transition function σ of $\mathscr{A}_\Delta$ can now be defined in a natural manner. Given the state $x_t(v) = f$ of node v in round t and a vector $\overline{a}_{t+1}(h_1, h_2, \ldots, h_\Delta)$ of messages received in round $t+1$, function σ gives a new state $x_{t+1}(v) = g$, where $g\colon \hat{\psi}_\Delta \to \{0,1,U\}$. Let us consider, for example, a subformula of the form $\langle\alpha\rangle\phi \in \hat{\psi}_\Delta$. If $f(\phi) = U$, the truth value of $\langle\alpha\rangle\phi$ cannot be determinted in round $t+1$, and hence $g(\langle\alpha\rangle\phi) = U$. If, on the other hand, $f(\phi) \in \{0,1\}$, then $g(\langle\alpha\rangle\phi) = h_i(\phi) \in \{0,1\}$, where h_i is the message sent by a neigbour dictated by the index α – here, the details depend on the choice of the logic $\mathscr{L}$.

If in round t we have $x_t(v) = f$ with $f(\psi_\Delta) \in \{0,1\}$, we move to the corresponding halting state $x_{t+1}(v) = f(\psi_\Delta)$. The end result is that $\mathscr{A}$ halts in round $\mathrm{md}(\psi_\Delta)+1$ and the output of $\mathscr{A}$ on node v is 1 if and only if $M_{a,b}(G,p,i), v \vDash \psi_\Delta$. It follows that $\mathbf{A} = (\mathscr{A}_1, \mathscr{A}_2, \ldots)$ solves the graph problem Π_Ψ.

In the other direction, we show that $\mathscr{L}$ simulates C on $\mathscr{M}_{a,b}$ by constructing for each graph problem $\Pi \in C$ a sequence $\Psi = (\psi_1, \psi_2, \ldots)$ of formulas of logic $\mathscr{L}$ such that Ψ defines a solution for Π. Suppose that $\mathbf{A} = (\mathscr{A}_1, \mathscr{A}_2, \ldots)$ is an algorithm that solves problem Π. We encode the states of nodes and messages sent during the execution of $\mathscr{A}_\Delta$ on (G,p,i) as follows – the formulas given are for the case $C = \mathsf{VV}(1)$; other cases are similar. Here $t \in \mathbb{N}$ is the running time of $\mathscr{A}_\Delta$.

- For each state $s \in S$ and round $r \in [t]$, we have formula $\phi_{s,r}$ that is true at point v iff $x_r(v) = s$.
- For each message $m \in M$, port number $j \in [\Delta]$ and $r \in [t]$, we have formula $\theta_{m,j,r}$ that is true at point v iff v sends message m to its port number j in round r.
- For each $m \in M$, $j, j' \in [\Delta]$ and $r \in [t]$, we have formula $\chi_{m,j,j',r}$ that is true at point v iff v receives message m from its port number j in round r and it was sent by a neighbour to port number j'.

Communication is implemented with the help of a modality $\langle\alpha\rangle$ by defining formula $\chi_{m,j,j',r}$ as follows:

$$\chi_{m,j,j',r} = \langle\alpha\rangle\theta_{m,j',r},$$

where $\alpha = (j, j')$. All the other formulas can be defined as Boolean combinations of already defined formulas in a recursive manner, in such a way that they have

the intended meaning. Finally, we can define $\psi_\Delta = \phi_{1,t}$. Now, the output of $\mathscr{A}_\Delta$ on a node v is 1 if and only if $M_{a,b}(G,p,i),v \vDash \psi_\Delta$. It follows that $\Psi = (\psi_1, \psi_2, \ldots)$ defines a solution to graph problem Π. Additionally, the modal depth $\mathrm{md}(\psi_\Delta)$ is equal to the running time t of $\mathscr{A}_\Delta$. □

3.2.3 Bisimulation in distributed computing

Now that we have Theorem 9 at our disposal, we can transfer the power of bisimulation from modal logic to distributed computing. This gives us a tool for showing that a given graph problem Π is not in one of the classes C that we defined in Section 3.1. Since bisimilarity implies indistinguishability by modal logic, we can use bisimulation to show that the corresponding modal logic cannot define a solution for Π, and via Theorem 9, that Π is indeed not in class C.

Noteworthily, we can apply this approach also for the non-constant-time versions of the problem classes, even though Theorem 9 only gives a correspondence between the constant-time versions of the classes and variants of modal logic. This is achieved by a simple trick, which we explain in the following corollary.

Corollary 10. *Let $G = (V,E) \in \mathscr{F}(\Delta)$ be a graph, p a port numbering for G, $i : V \to \Sigma$ an input labelling for G and $X \subseteq V$. Assume that Π is a graph problem such that for every $S \in \Pi(G,i)$ there exist $u,v \in X$ with $S(u) = 1$ and $S(v) = 0$.*

(1) If all nodes in X are bisimilar in the model $M_{+,+}(G,p,i)$, then $\Pi \notin \mathsf{VV}$.
(2) If all nodes in X are bisimilar in the model $M_{+,-}(G,p,i)$, then $\Pi \notin \mathsf{VB}$.
(3) If all nodes in X are bisimilar in the model $M_{-,-}(G,p,i)$, then $\Pi \notin \mathsf{SB}$.

Proof idea. Consider claim (1); the others are similar. Let $\mathscr{A}_\Delta \in \mathcal{VV}$ be any algorithm that halts on (G,p,i). Denote the running time of $\mathscr{A}$ on (G,p,i) by $t \in \mathbb{N}$. Now, define an algorithm $\mathscr{B}_\Delta$ as follows: $\mathscr{B}_\Delta$ simulates the execution of $\mathscr{A}_\Delta$, but maintains a counter that always stops the execution after t rounds. Algorithm $\mathscr{B}_\Delta$ clearly produces the same output on (G,p,i) as algorithm $\mathscr{A}_\Delta$, and furthermore, it has a constant running time.

Now, if $\Pi \in \mathsf{VV}$, we can assume to have a constant-time algorithm $\mathscr{B}_\Delta$ that solves Π on the fixed instance (G,p,i). It follows that we also have a formula ψ_Δ of MML that gives us a solution $S \in \Pi(G,i)$. Since all nodes in X are assumed to be bisimilar in $M_{+,+}(G,p,i)$, there cannot exist $u,v \in X$ with $S(u) = 1$ and $S(v) = 0$. Hence, we have $\Pi \notin \mathsf{VV}$. □

A similar result could be formulated also for the classes $\mathsf{VV_c}$, MV, SV and MB. However, we only need the cases given in Corollary 10 in this work.

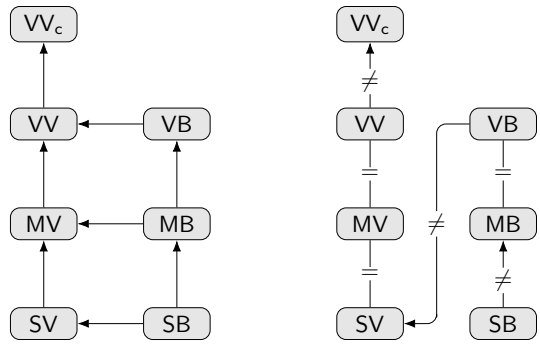


Figure 3.1. (a) Trivial containment relations between the problem classes. (b) The linear order obtained in our work.

3.3 Relationships between the models

To begin with, we note that the following containments follow directly from the definitions of the state machine classes:

$$\mathsf{SV} \subseteq \mathsf{MV} \subseteq \mathsf{VV} \subseteq \mathsf{VV_c}, \quad \mathsf{SB} \subseteq \mathsf{MB} \subseteq \mathsf{VB},$$

$$\mathsf{VB} \subseteq \mathsf{VV}, \quad \mathsf{MB} \subseteq \mathsf{MV}, \quad \mathsf{SB} \subseteq \mathsf{SV}.$$

Naturally, similar containments hold for the constant-time versions of the classes as well. These relations are depicted in Figure 3.1 (a).

On the other hand, some classes are seemingly orthogonal: for example, it is not clear whether either of VB and SV is contained in the other. In this section, we will present non-trivial separations and equivalences between the classes. Put together, they establish that actually, the classes form a linear order:

$$\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}.$$

That is, we achieve a complete classification of the relationships between the models. This is depicted in Figure 3.1 (b).

3.3.1 Equalities between the classes

Now, we are ready to go through the results that show certain classes of graph problems to be equal. We will only give brief descriptions of the proof ideas; the full proofs can be found in Publication I.

The following theorem is the most important technical contribution related to the relationships between the classes. It shows that outgoing port numbers can be used to reconstruct the multiplicities of incoming messages, at the cost of increasing the running time.

Theorem 11. *Let* $\mathbf{A} \in \mathbf{MV}$ *be an algorithm that solves a graph problem* Π *in time* $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. *Then there exists an algorithm* $\mathbf{B} \in \mathbf{SV}$ *that solves* Π *in time* T', *where* $T'(n, \Delta) = T(n, \Delta) + 2\Delta - 2$.

Proof idea. The proof is based on the idea that we first run algorithm $\mathscr{C}_\Delta \in \mathscr{SV}$ that uses *outgoing* port numbers to break symmetry between *incoming* messages for each node. After that, it is possible to simulate the original algorithm $\mathscr{A}_\Delta \in \mathscr{MV}$ by attaching this newly-gained symmetry-breaking information on each message.

For each $\Delta \in \mathbb{N}$, the algorithm $\mathscr{C}_\Delta$ runs for $2\Delta - 2$ rounds and essentially gathers all the information available in the (2Δ)-neighbourhood of a node: port numbers, node degrees and the graph structure that they reveal. The non-trivial part is showing that this is actually enough. To that end, we say that nodes u and w are a *pair of indistinguishable neighbours of* v *of order* k *in round* t if they are distinct neighbours of node v such that in the execution of $\mathscr{C}_\Delta$, they necessarily send the same message to v in each round from 1 to t, and furthermore, there are k distinct neighbours (possibly including u and w) that are in the same state with each other and with u and w in round $t - 1$.

In the crucial part of the proof, we show that if u and w are indistinguishable neighbours of v of order k in round t, then they are a pair of indistinguishable neighbours of v of order $k + 1$ in round $t - 2$. This is based on the observation that to make two neighbours u and w indistinguishable, they need to have k neighbours of their own that are in the same state as node v in round $t - 2$. However, due to the outgoing port numbers, those k neighbours of u and w each receive a message with distinct port number – but otherwise identical content – from u and w in round $t - 2$, which implies that v also has to receive the same k messages. As v also receives two entirely identical messages, the total number of messages, and thus neighbours, is at least $k + 1$.

With the above in mind, we can show by induction that no node can have indistinguishable neighbours in round $2\Delta - 1$. In other words, after executing the algorithm $\mathscr{C}_\Delta$ for $2\Delta - 2$ rounds, for each node v, all the neighbours of v are in such states that they are able to send different messages to v. Now, we can define the algorithm $\mathbf{B} = (\mathscr{B}_1, \mathscr{B}_2, \ldots) \in \mathbf{SV}$ as follows. For each Δ, we first execute the algorithm $\mathscr{C}_\Delta$. Then, we simulate the algorithm $\mathscr{A}_\Delta$, but amend each message sent by $\mathscr{A}_\Delta$ with the state produced by algorithm $\mathscr{C}_\Delta$. Each node v then receives $\deg(v)$ distinct messages on each round during the simulation and can thus reconstruct the multiset of messages. □

This immediately implies the following result.

Corollary 12. *We have the equalities* $\mathsf{SV} = \mathsf{MV}$ *and* $\mathsf{SV}(1) = \mathsf{MV}(1)$.

Next, we take a look at the case of recovering incoming port numbers from a multiset of messages. Note that the proof we have for Theorem 11 could be used to show also Theorem 13 (a): the symmetry-breaking information created by algorithm $\mathbf{C} \in \mathbf{SV} \subseteq \mathbf{MV}$ could be used to order incoming messages in a consistent

manner and hence simulate incoming port numbers that are available to an algorithm in **VV**. However, a different approach allows us to avoid the increase in running time, and moreover, the same approach proves also Theorem 13 (b).

Theorem 13. *(a) Let* $\mathbf{A} \in \mathbf{VV}$ *be an algorithm that solves a graph problem* Π *in time* $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. *Then there exists an algorithm* $\mathbf{B} \in \mathbf{MV}$ *that solves* Π *in time* T.

(b) Let $\mathbf{A} \in \mathbf{VB}$ *be an algorithm that solves a graph problem* Π *in time* $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. *Then there exists an algorithm* $\mathbf{B} \in \mathbf{MB}$ *that solves* Π *in time* T.

Proof idea. Given an algorithm $\mathscr{A}_\Delta \in \mathcal{VV}$ and a graph G with port numbering p and input labeling i, we construct an algorithm $\mathscr{B}_\Delta \in \mathcal{MV}$ that simulates the execution of $\mathscr{A}_\Delta$ on (G, p', i), where p' is another port numbering for G.

The simulation is achieved by augmenting each message sent by $\mathscr{A}_\Delta$ with the full communication history. On each node v and for each $j \in [\deg(v)]$, algorithm $\mathscr{B}_\Delta$ stores all messages sent to port (v, j) in $\mathscr{A}_\Delta$, and on each round, attaches the full message history to the message sent. We fix some linear order for the message set M of $\mathscr{A}_\Delta$. On each round, the received message histories are stored and ordered lexicographically by $\mathscr{B}_\Delta$ to construct a vector.

The lexicographical ordering essentially creates incoming port numbers – denote this virtual port numbering by p'. Note that incoming port numbers given by p' are possibly different from those given by the actual port numbering p. This is not an issue, since by definition, $\mathscr{A}_\Delta$ produces a valid solution for Π, given any port numbering for G. Hence, the solution produced by $\mathscr{B}_\Delta$ on (G, p, i) is also valid.

The method for simulating an algorithm $\mathscr{A}_\Delta \in \mathcal{VB}$ by an algorithm $\mathscr{B}_\Delta \in \mathcal{MV}$ is the same as above; in this case, each node just broadcasts the same message to each neighbour. □

We note that the proof idea for Theorem 13 first appeared implicitly in the work of Åstrand and Suomela [3]. An explicit, more detailed proof was given in Publication I, but the presentation contained a minor defect: it was not taken into account that in the simulated algorithm $\mathscr{A}_\Delta$, some nodes may halt earlier than others – and hence start sending the dummy message $\emptyset$. This was remedied by a proof given in the author's master's thesis [27].

Again, we immediately obtain the following result.

Corollary 14. *We have the equalities* $\mathsf{MV} = \mathsf{VV}$, $\mathsf{MV}(1) = \mathsf{VV}(1)$, $\mathsf{MB} = \mathsf{VB}$ *and* $\mathsf{MB}(1) = \mathsf{VB}(1)$.

When we put together Corollaries 12 and 14, as well as the trivial containment relations, we obtain

$$\mathsf{SB} \subseteq \mathsf{MB} = \mathsf{VB} \subseteq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subseteq \mathsf{VV_c}.$$

Thus, the only thing left is to show that the subset relations are proper.

3.3.2 Separations between the classes

Next, we present the results of Publication I that separate one class of graph problems from another. Many of the separation results are essentially known from the earlier work of Yamashita and Kameda [36], but their proof techniques are different. The purpose of our work is to demonstrate the usefulness of the connections to modal logic, and in particular bisimulation. We note that none of our separations rely on local node inputs.

To separate two classes, it is enough to construct a graph problem that is in one class but is not in the other. Let us start with an easy one.

Theorem 15. *There exists a graph problem* $\Pi \in \mathsf{SV}(1) \setminus \mathsf{VB}$.

Proof idea. Let Π be the problem of selecting a node in a star graph: If $G = (V, E)$ is a star, a binary labelling $S : V \to \{0,1\}$ is a valid solution if and only if we have $S(v) = 1$ for exactly one leaf node v and $S(v) = 0$ for all other nodes v of G. For other graphs G, any $S : V \to \{0,1\}$ is a valid solution.

This is solved by a constant-time algorithm $\mathbf{A} \in \mathbf{SV}$: Each node sends message 1 to only one neighbour and 0 to all others. Then, the node of degree 1 that received message 1 outputs 1, others output 0.

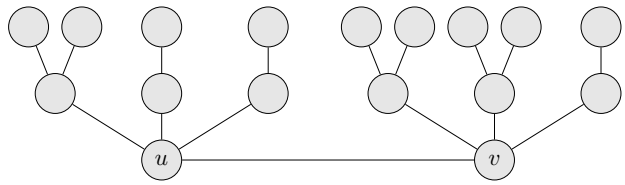
To see that $\Pi \notin \mathsf{VB}$, consider a star graph $G = (V, E)$ and let $X \subseteq V$ consist of all the leaf nodes. For any port numbering p for G, all nodes in X are bisimilar in the model $M_{+,-}(G, p, i)$, and hence Corollary 10 implies the result. □

Theorem 16. *There exists a graph problem* $\Pi \in \mathsf{MB}(1) \setminus \mathsf{SB}$.

Proof idea. Let Π be the problem of detecting whether the number of neighbours of an odd degree is odd.

Consider the following algorithm $\mathbf{A} \in \mathbf{MB}$: Each node v with an odd degree $\deg(v)$ broadcasts message 1 to its neighbours, while all other nodes broadcast 0. Then, each node counts the number of messages 1 it received, and outputs 1 if and only if this number was odd. This shows that $\Pi \in \mathsf{MB}(1)$.

To see that $\Pi \notin \mathsf{SB}$, consider the graph G pictured below.

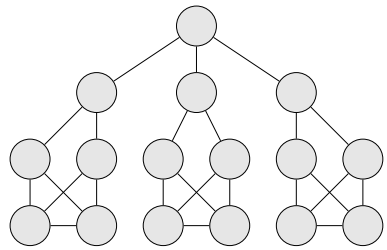


Let $X = \{u, v\}$. The nodes u and v are bisimilar in $M_{-,-}(G, p, i)$ for any port numbering p, but they are required to produce different output. Hence, Corollary 10 applies. □

Theorem 17. *There exists a graph problem* $\Pi \in \mathsf{VV_c}(1) \setminus \mathsf{VV}$.

Proof idea. This is more involved than the previous two separation results. Briefly, the idea is to show that there exists a graph G that (1) has a port numbering p such that in $M_{+,+}(G,p,i)$, all nodes are bisimilar with each other, but that (2) does not have a *consistent* port numbering with the same property. In such a graph G, an algorithm $\mathcal{A}_\Delta \in \mathcal{VV}$ can produce different output in two nodes if and only if the port numbering is guaranteed to be consistent. This is enough to separate $\mathsf{VV_c}(1)$ from VV.

The below graph is an example of a graph with the desired property.



To show the existence of a port numbering p that makes all nodes bisimilar, it turns out to be enough that the graph is regular. On the other hand, the crucial property that makes it impossible for this port numbering p to be consistent is the lack of a perfect matching in the graph. □

When we put all the equivalence and separation results together, we obtain the following result, which fully characterises the relations between the classes. Together with Theorem 9, this is the main result of Publication I.

Corollary 18. $\mathsf{SB} \subsetneq \mathsf{MB} = \mathsf{VB} \subsetneq \mathsf{SV} = \mathsf{MV} = \mathsf{VV} \subsetneq \mathsf{VV_c}$.

3.3.3 Lower bounds for simulating MV in SV

Recall that Theorem 11 established the equivalence of classes SV and MV, as well as their constant-time variants, but at the cost of increasing the running time by $2\Delta - 2$ rounds. In Publication I, it was left open whether the overhead in running time is really necessary.

The results of Publication II answer this in the affirmative. The first result is a tight lower bound for the simulation overhead:

Theorem 19. *For each $\Delta \geq 2$, there exists a graph $G = (V,E) \in \mathcal{F}(\Delta)$, a port numbering p for G and nodes $v,u,w \in V$ such that when executing any algorithm $\mathcal{A} \in \mathcal{SV}$ on (G,p), node v receives identical messages from its neighbours u and w in rounds $1,2,\ldots,2\Delta-2$.*

This is an exact counterpart to Theorem 11 in the sense of showing that the symmetry-breaking procedure employed in the proof of Theorem 11 cannot be made any faster.

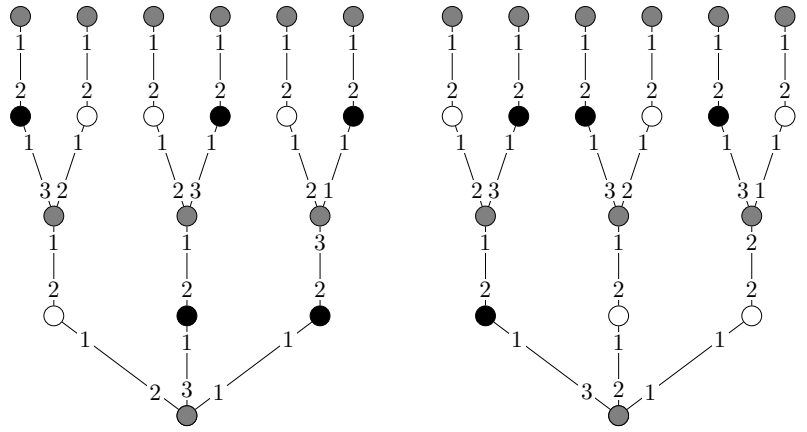


Figure 3.2. An algorithm in $\mathcal{MV}$ can distinguish the bottommost nodes of graphs (a) and (b) in one communication round, while any algorithm in $\mathcal{SV}$ needs at least three communication rounds.

The second result of Publication II extends the linear-in-Δ lower bound to graph problems as follows.

Theorem 20. *There exists a graph problem* Π *that can be solved in just one communication round by an algorithm in* **MV** *but that requires at least time* T*, where* $T(n,\Delta) \geq \Delta$ *for each odd* $\Delta \in \mathbb{N}_+$ *and* $T(n,\Delta) \geq \Delta - 1$ *for each even* $\Delta \in \mathbb{N}_+$*, when solved by an algorithm in* **SV**.

The proof technique behind both of these results is that we construct a family of graphs with very intricate port numberings. These port numberings make it as hard as possible for algorithms to distinguish nodes from each other. Figure 3.2 illustrates Theorem 20 by giving the construction for $\Delta = 3$.

Again, we use bisimulation to prove that certain nodes are indistinguishable by distributed algorithms in certain classes. However, in this case, we use the finite approximation we call r-bisimulation. Thus, Publication II further demonstrates the power of bisimulation – and logical tools in general – in the realm of distributed computing.

4. Space and Time in Distributed Computing

In this chapter, we consider the relationship between space and time complexity in distributed computing. The question we ask is the following: is there a graph problem that can be solved in constant space but that requires a non-constant amount of time? A constant time complexity trivially implies a constant number of states visited during the execution, but the other direction is more interesting. We answer the question affirmatively: we construct a non-trivial graph problem that can be solved in constant space in a very limited model of computation, while requiring a linear amount of communication rounds even in the class of path and cycle graphs and in a much stronger model of computation. In Section 4.1 we outline the aspects that make our question challenging, in Section 4.2 we give definitions needed later in this chapter, and finally, in Section 4.3, we present our graph problem and an algorithm for solving it.

4.1 Challenges

At first, it may seem like it is straightforward to obtain a constant space complexity and non-constant time complexity. After all, even if every node is executing a constant-space algorithm, the total amount of space available in the distributed system is linear. This makes it possible, for example, to simulate a linear-space Turing machine in a path graph and spend an exponential amount of time.

However, our question is not about wasting time – it is about the *existence of a graph problem* with certain time and space complexities. Additionally, we do not want to assume that the input graph is for example a path – instead, we want to have a graph problem *without any promises*. Recall that in our framework of distributed computing, we require each node to halt execution after a finite amount of communication rounds. Even though this is easy to achieve in for instance path graphs, *cycle* graphs are a different matter.

Consider the t-neighbourhood of a node v in a graph G. If all nodes in this t-neighbourhood have a degree of 2, it is impossible for node v to tell whether G is a path or a cycle graph – or something completely different – based on this neighbourhood. In a constant amount of space, node v can remember its

t-neighbourhood only for some fixed $t \in \mathbb{N}$. Still, node v is required to terminate after a finite amount of time. It might seem natural to think that any graph problem solvable in constant space in this kind of setting would also be solvable in constant time. However, it turns out that this is not true.

4.2 Preliminaries

Models of computation. The model of computation that we mainly consider in this chapter is the weakest variant of the port-numbering model that we defined in Section 3.1 – namely the SB model. Recall that a distributed state machine in class $\mathcal{SB}$ receives messages in a *set* and *broadcasts* the same message to each neighbour. We also discuss the LOCAL model defined in Section 2.3.4. Our upper bound will be given for the SB model and our lower bound for the LOCAL model – note that while SB is a very limited model, this actually makes our result stronger.

Thue–Morse sequence. The central notion behind our curious graph problem is the *Thue–Morse* sequence [1], and in particular, the recursive definition of its prefixes.

Definition 21. The *Thue–Morse* sequence is the infinite binary sequence (t_i) where $t_0 = 0$, and for each $i \in \mathbb{N}$, $t_{2i} = t_i$ and $t_{2i+1} = 1 - t_i$.

Thus, the beginning of the Thue–Morse sequence is

$$01101001100101101001\ldots$$

Definition 22. Let $T_0 = 0$. For each $i \in \mathbb{N}_+$, let T_i be obtained from T_{i-1} by substituting each occurrence of 0 with 01 and each occurrence of 1 with 10. We call T_i the *Thue–Morse word of length* 2^i.

It can be seen by an induction argument that for each $i \in \mathbb{N}$, the word T_i is the prefix of length 2^i of the Thue–Morse sequence.

4.3 Constant space and linear time in path and cycle graphs

The main result of Publication III is the following.

Theorem 23. *There exists a decision graph problem* Π_{TM} *with* $O(1)$ *space complexity and* $\Theta(n)$ *time complexity in the class of path and cycle graphs in the* SB *model. Furthermore, the time complexity of* Π_{TM} *is* $\Theta(n)$ *also in the* LOCAL *model.*

To define the graph problem Π_{TM}, we need to adapt the definition of the Thue–Morse sequence slightly.

Definition 24. Define a set of words over $\{0,1,\_\}$ recursively as follows:

(1) _0_ is *valid*.

(2) If X is valid and Y is obtained from X by applying the substitutions $0 \mapsto 0\_1\_1\_0$ and $1 \mapsto 1\_0\_0\_1$ to each occurrence of 0 and 1 in X, then Y is *valid*.

Observe that one can obtain valid words from the prefixes $T_{2i}, i \in \mathbb{N}$, of the Thue–Morse sequence by inserting an underscore at the beginning, in between each symbol and at the end.

Definition 25. Let $\Sigma = \{0,1,2\} \times \{0,1,\_\}$ be a set of local input labels and let and $\Gamma = \{\mathsf{yes}, \mathsf{no}\}$ be a set of local output labels. Given a labelling $i \colon V \to \Sigma$ for a graph $G = (V,E)$, we write $i(v) = (i_1(v), i_2(v))$ for each $v \in V$. The decision graph problem Π_{TM} is now defined as follows. An instance (G,i) is a yes-instance of Π_{TM} if and only if the following conditions hold:

(1) The graph $G = (V,E)$ is a path graph.

(2) For each $v \in V$ with $\deg(v) = 2$, we have

$$\{i_1(u) : u = v \text{ or } u \in N(v)\} = \{0,1,2\}.$$

That is, by defining $v < u$ iff $i_1(v) + 1 = i_1(u) \bmod 3$, we get a consistent orientation for the path.

(3) If we write $V = \{v_1, v_2, \ldots, v_n\}$ such that $\{v_i, v_{i+1}\} \in E$ and $v_i < v_{i+1}$ for each $i \in \{1,2,\ldots,n-1\}$, the word

$$i_2(v_1) i_2(v_2) \ldots i_2(v_n)$$

defined over $\{0,1,\_\}$ by the second parts $i_2(\cdot)$ of the input labels is valid.

4.3.1 The algorithm

We will now give an overview of the constant-space algorithm that solves problem Π_{TM}. The detailed definition can be found in Publication III.

Part I. We divide the execution of the algorithm into three parts. Consider an instance (G,i) of problem Π_{TM}. In the first part, each node v of G collects the input labels of its neighbours and checks the following:

- Its own degree $\deg(v)$ is either 1 or 2.
- If $\deg(v) = 2$, then $\{i_1(u) : u = v \text{ or } u \in N(v)\} = \{0,1,2\}$.

Recall that we assume graphs to be finite and connected. If both of the above conditions hold for every node v, it follows that (1) G is either a path or a cycle graph and (2) we can consistently orient the edges of G by orienting each edge

$e = \{v, u\}$ from v to u iff $i_1(v) + 1 = i_1(u) \bmod 3$ (unless G is a path of length 2). In this case, we call v the *left neighbour* of u and u the *right neighbour* of v.

From now on, we can utilise the orientation of the edges to aid communication. By attaching the label $i_1(v)$ to each message sent by node v, receiving nodes can distinguish between messages received from left and right. Similarly, each message sent can be equipped with the label $i_1(u)$ if the intended recipient is u. Essentially, this causes our algorithm in class $\mathscr{SB}$ to have capabilities that are available in the standard port-numbering model.

Part II. Then, in the second part, each node v of G checks that the word defined by the labels $i_2(\cdot)$ is *locally valid*:

- If $\deg(v) = 1$ and the neighbour of v is u, then $i_2(v) = \_$ and $i_2(u) = 0$.
- If $\deg(v) = 2$ and the left and right neighbour of v are u_1 and u_2, respectively, then the word $i_2(u_1)i_2(v)i_2(u_2)$ is in the set {0_0, 1_1, 0_1, 1_0, _0_, _1_}.

These conditions guarantee that in the input word (note that the word can be circular), symbols in $\{0,1\}$ are separated from each other by exactly one underscore, and that in the case of a path graph, the input word begins and ends with the underscore symbol.

Part III. In the final part, the validity of the input word is verified in a recursive, non-local manner. This is done in several *phases*. Each node v holds a *current symbol* $c(v)$. Initially, $c(v) = i_2(v)$. From a global viewpoint, what happens to the word consisting of the current symbols is the following. During each phase, for $i \in \mathbb{N}_+$, each subword

$$\_0^i\_1^i\_1^i\_0^i\_1^i\_0^i\_0^i\_1^i\_$$

is substituted with

$$\_0^i00^i00^i00^i\_1^i11^i11^i11^i\_,$$

and similarly, each subword

$$\_1^i\_0^i\_0^i\_1^i\_0^i\_1^i\_1^i\_0^i\_$$

is substituted with

$$\_1^i11^i11^i11^i\_0^i00^i00^i00^i\_.$$

Compare this to Definition 24. If the input word is a valid word, then on each phase, we are essentially going backwards in the recursive definition of valid words – with the difference that each symbol 0 or 1 is represented by a sequence of 0s or 1s, respectively. We call such a word a *padded* valid word. In particular, the substitution is unambiguous: the new current symbol of a node does not depend on where we choose to match the given subword. This means that ultimately, the current word will be either

$$\_0^i\_ \quad \text{or} \quad \_0^i\_1^i\_1^i\_0^i\_$$

for some $i \in \mathbb{N}_+$. At that point, the algorithm halts and accepts the input. On the other hand, if the input word is not valid – regardless of whether of graph is a path or a cycle – we can show that eventually, the current word contains a subword that cannot be substituted according to the above rules, or alternatively, the substitution would be ambiguous. In that case, the algorithm halts and rejects.

Here the essential observation is the following. When gathering the current symbols in the r-neighbourhood of a node v, we do not need to store the entire word – instead, we can collapse sequences of 0s and 1s. In other words, we only need to check whether the pattern

$$\_0+\_1+\_1+\_0+\_1+\_0+\_0+\_1+\_$$

or the pattern

$$\_1+\_0+\_0+\_1+\_0+\_1+\_1+\_0+\_,$$

where $a+$ matches to a sequence of one or more symbols a, matches the neighbourhood of node v. Thus, even though the number of communication rounds needed increases after each phase, the amount of space needed remains constant.

Details. We implement the word substitution locally as follows. Each node v maintains two *buffers*, one for its left neighbourhood and another for its right neighbourhood. On each round, the left buffer is sent to the right neighbour and vice versa. As v gathers its neighbourhood, it collapses subsequent 0s and 1s on the fly – the current symbol $c(v)$ of v is appended to the end of the buffer received from the left neighbour if and only if the last symbol of the buffer is different from $c(v)$, and similarly for the buffer received from the right. When both buffers of v contain either 8 occurrences of the separator _ or cannot be grown further due to reaching the end of a path graph, node v is finished with gathering its neighbourhood and performs the substitution explained above.

Correctness. For our algorithm to be well-defined and work correctly in Part III, several conditions need to hold. First, we need the phases to happen in synchrony: each node starts a new phase in the same round. Related to this, the length of each block of subsequent 0s and 1s needs to stay equal. Second, the local substitutions made by nodes need to result in the correct behaviour globally. On one hand, if the current word is a padded valid word of length 2^k, the new word is a padded valid word of length 2^{k-2}. On the other hand, if the current word is not a padded valid word, the new word must also not be one, so that the substitution eventually fails and the input gets rejected. While these properties are not deep in any sense, the detailed proofs are somewhat technical.

Time complexity. Consider then the time complexity of our algorithm. Each phase takes an amount of communication rounds linear in the number of the phase, and a logarithmic number of phases is enough. This results in a total running time of $O(n)$. We also get a matching lower bound. It is clear that the time complexity of Π_{TM} is in $\Omega(n)$: a node at the end of a path has to receive

information from the other end of the path to be able to verify that the instance is a yes-instance. This holds also in much stronger models of computation, for example in the LOCAL model. We have now concluded our overview of the proof of Theorem 23.

5. Distributed Time Complexity Classes

In this section, we introduce a new general technique for constructing LCL problems with specific time complexities. We use the technique to construct infinitely many problems with complexities not previously known to exist. The new complexities can be divided into two distinct categories: the high and the low. In the high category, we obtain for example $\Theta(\log^\alpha n)$ for any $\alpha \geq 1$, $2^{\Theta(\log^\alpha n)}$ for any $\alpha \leq 1$ and $\Theta(n^\alpha)$ for any $\alpha < 1/2$. On the other hand, the low category contains complexities such as $\Theta(\log^\alpha \log^* n)$ for any $\alpha \geq 1$, $2^{\Theta(\log^\alpha \log^* n)}$ for any $\alpha \leq 1$, and $\Theta((\log^* n)^\alpha)$ for any $\alpha \leq 1$. The constant α is a positive rational number.

The high-level idea of our technique is as follows. We define a centralised model of computation we call a *link machine* model. A link machine consists of a constant number of registers that can store unbounded positive natural numbers, as well as a finite program that operates on the registers. We define a way for link machines to compute functions $g\colon \mathbb{N} \to \mathbb{N}$ – we say that machine M has *growth* g. Then, given a link machine M with growth g and an LCL problem Π with time complexity T on directed cycles, we construct a new LCL problem Π_M whose time complexity is smaller than T, by an amount that is controlled by function g.

A lot of care is needed to ensure that the resulting problem Π_M is actually an LCL problem and has the desired complexity. We will proceed by defining the concept of link machines in Section 5.1 and the method for constructing new graph problems in Section 5.2. Finally, we put the ingredients together to obtain new complexities in Section 5.3.

5.1 Link machines

Definition and growth. Given a set of *register* labels $\{r_1, r_2, \ldots, r_k\}$, we define a *program* P to be a sequence $(i_1, i_2, \ldots, i_p)$ of instructions that operate on k registers containing natural numbers. Each instruction i_j is one of the following, for some register labels a, b and c, and $s \in \mathbb{N}_+$:

- Addition: $a \leftarrow b + c$.
- Copy: $a \leftarrow b$.
- Reset: $a \leftarrow 1$.
- Conditional execution: if $a = b$ (or $a \neq b$), skip the next s instructions.

Now, a *link machine* M consists of a finite set of register labels and a program P that operates on the corresponding registers. The link machine M can have designated *input* and *output registers* x and y among its k registers.

An *execution* of a link machine M is a single run through its program P. Before the first execution, each register is assumed to contain value 1. On each execution, the program P modifies the register values according to its instructions in the obvious way. For each $\ell \in \mathbb{N}$, we denote the final value of each register a after ℓ executions by $a(\ell)$. Additionally, we denote by $a(\ell, j)$ the value of register a after executing the machine $\ell - 1$ times and then executing the first j instructions $i_1, i_2, \ldots, i_j$ of P.

A link machine M with registers $r_1, r_2, \ldots, r_k$ has *growth* $g \colon \mathbb{N} \to \mathbb{N}$ if

$$g(\ell) = \max\{r_i(\ell) : i \in [k]\} \quad \text{for all } \ell \in \mathbb{N},$$

that is, the growth of M keeps track of the maximum register value after each execution. For a growth g, we write $g^{-1} \colon \mathbb{N} \to \mathbb{N}$ to denote the function given by

$$g^{-1}(\ell) = \min\{m \in \mathbb{N} : g(m) \geq \ell\} \quad \text{for all } \ell \in \mathbb{N}.$$

We note that even though register values are unbounded, the growth of a link machine is not able to exceed $2^{O(\ell)}$.

Composition. Let M_1 and M_2 be link machines with programs $P_1 = (i_1^1, i_2^1, \ldots, i_p^1)$ and $P_2 = (i_1^2, i_2^2, \ldots, i_q^2)$, respectively. Assume that M_1 has an output register y and M_2 has an input register x. By relabelling the registers, if necessary, we can assume that M_1 and M_2 do not share any register labels. Now, we define the *composition* $P_2 \circ P_1$ as follows:

$$P_2 \circ P_1 = (i_1^1, i_2^1, \ldots, i_p^1, x \leftarrow y, i_1^2, i_2^2, \ldots, i_q^2).$$

The corresponding new machine $M_2 \circ M_1$ consists of the union of the register labels of M_1 and M_2 and the program $P_2 \circ P_1$. If M_1 has input register x', $M_2 \circ M_1$ also has input register x', and if M_2 has output register y', so has $M_2 \circ M_1$. Now, the composition $M_i \circ \cdots \circ M_2 \circ M_1$ of multiple link machines can be defined in a natural way.

Basic building blocks. To achieve the different growths that we need to establish our new time complexities for LCL problems, we start by constructing a collection of basic link machines. We then use these machines to obtain the desired growths by taking compositions.

The basic link machine programs that we introduce in Publication IV are the following.

- COUNT: output register $y = \ell$ and growth ℓ,
- ROOT'_k for each $k \in \mathbb{N}_+$: output register y in $\Theta(\ell^{1/k})$ and growth in $\Theta(\ell^{1/k})$,
- ROOT_k for each $k \in \mathbb{N}_+$: input register x, output register y in $\Theta(x^{1/k})$ and growth in $\Theta(x)$,
- POW_k for each $k \in \mathbb{N}_+$: input register x, output register y in $\Theta(x^k)$ and growth in $\Theta(x^k)$,
- EXP: input register x, output register y in $2^{\Theta(x)}$ and growth in $2^{\Theta(x)}$,
- LOG: input register x, output register y in $\Theta(\log x)$ and growth in $\Theta(x)$.

Then, we use the above building blocks to get the following more complicated programs and growths.

- $\text{POW}_p \circ \text{ROOT}'_q$ for each $p, q \in \mathbb{N}_+$: growth in $\Theta(\ell^{p/q})$,
- $\text{EXP} \circ \text{POW}_q \circ \text{ROOT}'_p$ for each $p, q \in \mathbb{N}_+$, $p \geq q$: growth in $2^{\Theta(\ell^{q/p})}$,
- $\text{EXP} \circ \text{POW}_q \circ \text{ROOT}_p \circ \text{LOG} \circ \text{COUNT}$ for each $p, q \in \mathbb{N}_+$, $p \geq q$: growth in $2^{\Theta(\log^{q/p} \ell)}$.

Many more growth complexities could be obtained by link machines, but the above are enough for our purposes.

5.2 From link machines to graph problems

In this section, we explain the basic idea on how to encode link machines into locally checkable graphs, and furthermore, how to construct graph problems based on these graphs.

Link machine encoding graphs. Let M be a link machine with k registers, with a program of length p and with growth g such that $g\colon \mathbb{N} \to \mathbb{N}$ is non-decreasing and is in $\omega(1)$ as well as in $2^{O(n)}$. We will next define *link machine encoding graphs* for M. They are directed grid graphs, with a set of labels for each node and incident edge. For each $h \in \mathbb{N}_+$, we define a graph as follows.

First, we have a 2-dimensional $n \times h$ grid graph, where $n = 3g(h)$. We denote by (x, ℓ) the node on the xth column and the ℓth row; here $x \in [n]$ and $\ell \in [h]$. The grid wraps around along the horizontal axis but not along the vertical axis. In other words, there are edges $((n, \ell), (1, \ell))$ for each $\ell \in [h]$. Each node (x, ℓ) associates one of the labels U, D, L or R to each of its incident edges e depending of whether the other endpoint of e is on row $\ell + 1$, row $\ell - 1$, on column $x - 1$ or on column $x + 1$, respectively.

Then, the base grid described above is augmented with *link edges* that encode the execution of the machine M. Given a node (x, ℓ), a *link edge of length s* from (x, ℓ) is an edge $\big((x, \ell), (x + s \bmod n, \ell)\big)$. That is, link edges are horizontal and

create shortcuts in the grid structure. For each $\ell \in [h]$, each register r of M and each $j \in [0, p]$, there is a link edge of length $r(\ell, j)$ from all nodes on level ℓ. Each node (x, ℓ) associates the label (r, j) to the link edge of length $r(\ell, j)$ from it. Note that the length $r(\ell, j)$ can be equal for multiple different registers r – in that case there are multiple labels (r, j) for a single edge.

Finally, each node (x, ℓ) is labelled with an input value $i(x, \ell) \in \{0, 1\}$.

Local checkability. The crucial property that we need for our graph family is that graphs in it are *locally checkable*. This is to say that there is a constant-time algorithm in the LOCAL model such that given a graph from the family, all nodes accept, and given any other graph, at least one node rejects. It turns out that to make link machine encoding graphs locally checkable, we need to relax the definition slightly.

First, let us specify a collection of constraints that each node v can check in four rounds of communication. We write $v(L_1, L_2, \ldots, L_k)$ to denote the node that can be reached from the current node by following edges labelled with $L_1, L_2, \ldots, L_k$. The constraints are as follows.

(1) Labels L and R, and at least one of U and D, are present on incident edges. No label occurs twice and no edge has two different direction labels.

(2) Each edge labelled with a direction label has the opposite label at the other end. If v has an edge labelled with D, then $v(\mathsf{D}, \mathsf{R}, \mathsf{U}) = v(\mathsf{R})$. For each $L \in \{\mathsf{U}, \mathsf{D}\}$, if v does not have an edge labelled with L, then neither $v(\mathsf{L})$ nor $v(\mathsf{R})$ has an edge labelled with L.

(3) Link edges encode correctly initialised register values. That is, edges with label $(r, 0)$ either encode value 1 (if v does not have an edge labelled with D) or encode a value copied from the row below (otherwise).

(4) Link edges encode correctly the execution of program P. When going through the instructions of P one by one, the corresponding link edges point to nodes in keeping with the instructions.

(5) If v does not have an edge labelled with U, the link edges corresponding to the maximum register of M form 3-cycles.

Link machine encoding graphs satisfy the above constraints. The converse is not necessarily true: if register values exceed the width of the grid, the correspondence between edge lenghts and register values does not hold in the top part of the grid. We say that a graph is an *extended link machine encoding graph* if it satisfies the above constraints and has at least one node without an edge labelled with D.

LCL problems. We are now ready to define our LCL problem Π_M based on an LCL problem Π on directed cycle graphs and a link machine M. In our work, Π is either 3-colouring, in which case Π has time complexity $\Theta(\log^* n)$ or a variant

of 2-colouring that is always solvable, in which case the time complexity of Π is $\Theta(n)$. The growth of M is assumed to be non-decreasing, in $\omega(1)$ and in $2^{O(n)}$.

The output labels of problem Π_M are the following:

- output labels of the problem Π,
- an error label E,
- an error pointer $\mathsf{EP}(L, c_1, c_2)$, where $L \in \{\mathsf{R}, \mathsf{U}\}$ and $c_1, c_2 \in \{0, 1\}$,
- an empty output ϵ.

Valid solutions for Π_M are defined as follows.

(1) Label E is a valid output at node v if and only if the input labelling does not satisfy the constraints of extended link machine encoding graphs at v or at a neighbour of v. In this case, E is the only valid label.

(2) An output label of Π is valid at node v only if v does not have an incident edge with label D and the local constraints of problem Π are satisfied.

(3) Label ϵ is a valid output at node v only if v has an incident edge with label D.

(4) Label $\mathsf{EP}(L, c_1, c_2)$ is a valid output at node v if the following holds:

- If v has an incident edge with label D, then $L = \mathsf{U}$.
- The neighbour $v(L)$ of v outputs either E or $\mathsf{EP}(L', c_1', c_2')$.
- If $L = \mathsf{R}$ and $v(\mathsf{R})$ outputs $\mathsf{EP}(\mathsf{R}, c_1', c_2')$, then $c_1 \neq c_1'$.
- If $L = \mathsf{U}$ and $v(\mathsf{U})$ outputs $\mathsf{EP}(\mathsf{U}, c_1', c_2')$, then $c_2 = c_2'$.
- If $L = \mathsf{U}$ and $v(\mathsf{U})$ outputs E, then $c_2 = i(v(\mathsf{U}))$.

Since the above conditions are locally checkable, Π_M is an LCL problem. The intuition behind Π_M is as follows. An algorithm has basically two options: solve the original problem Π on the bottom cycle or point out an error in the graph structure.

The former is made faster by the link edges in a way specified by the link machine M, while the latter is made slower by our definition of Π_M. To point out an error, an algorithm has to produce a chain of pointers that first goes horizontally right and then vertically up, the horizontal part has to be 2-coloured, and the vertical part has to reproduce the input $i(v)$ given to the node at the end of the chain that sees an error locally.

5.3 New complexities for the LOCAL model

Let us now take a look at the time complexities of the new LCL problems Π_M we constructed in Section 5.2. Let $T\colon \mathbb{N} \to \mathbb{N}$ be the time complexity of problem Π and let $g\colon \mathbb{N} \to \mathbb{N}$ be the growth of link machine M. Define $f\colon \mathbb{N} \to \mathbb{N}$ by setting $f(k) = kg(k)$. Given $n \in \mathbb{N}_+$, let $\hat{n}$ be the smallest natural number for which $n \leq \hat{n} f^{-1}\big(T(\hat{n})\big)$ holds. The intuition behind this is that the hardest instances

of size n of graph problems Π_M are grids that have width roughly $\hat{n}$ and height roughly $f^{-1}\big(T(\hat{n})\big)$.

Now, the following theorems relate the complexity of Π_M to the complexity of Π and to the growth of M.

Theorem 26. *The time complexity of problem Π_M is in $O\big(f^{-1}\big(T(\hat{n})\big)\big)$.*

Proof idea. The crucial thing here is that the link edges of a link machine encoding graph provide shortcuts. If we start from a node v on the bottom cycle, go k steps up, then k steps right along link edges, and finally k steps down, we reach a node that is at distance $f(k) = kg(k)$ from v along the bottom cycle. Note that we only took $O(k)$ steps. By using a similar procedure, we can reach any node u that is at distance k from v along the bottom cycle by taking only $O\big(f^{-1}(k)\big)$ steps.

On the other hand, if the input graph is not a valid link machine encoding graph, this can also be detected in $O\big(f^{-1}\big(T(\hat{n})\big)\big)$ rounds. Hence we can always solve Π_M fast enough by either solving problem Π on the bottom cycle or by pointing our an error in the graph. □

Theorem 27. *The time complexity of problem Π_M is in $\Omega\big(f^{-1}\big(T(\hat{n})\big)\big)$.*

Proof idea. The worst-case instances of Π_M are graphs that we call *truncated* link machine encoding graphs. They are obtained from valid link machine encoding graphs by removing a suitable number of rows from the top of the grid. Then, each algorithm has to either solve problem Π on the bottom cycle or to point out an error.

We can show the former to require $\Omega\big(f^{-1}\big(T(\hat{n})\big)\big)$ rounds by a simulation argument: a faster algorithm for Π_M could be simulated to obtain a faster algorithm for Π on cycle graphs. The latter also requires $\Omega\big(f^{-1}\big(T(\hat{n})\big)\big)$ rounds due to this being the distance from the bottom cycle to the top where the error is located. Finally, an algorithm could try to produce a cycle of error pointers, all of which are pointing right, but this is made hard by the requirement that the chain of pointers is 2-coloured. □

We are now ready to state the end result of this technique.

Theorem 28. *Let r, s, p and q be non-negative natural numbers such that $r/s < 1/2$ and $q/p \le 1$. Then there exist* LCL *problems with time complexities*

(1) $\Theta\big(n^{r/s}\big)$,
(2) $\Theta\big((\log^* n)^{q/p}\big)$,
(3) $\Theta(\log^{p/q} n)$,
(4) $\Theta(\log^{p/q} \log^* n)$,
(5) $2^{\Theta(\log^{q/p} n)}$,
(6) $2^{\Theta(\log^{q/p} \log^* n)}$.

Proof idea. Let us consider claim (1). Let M be a link machine with program $\mathrm{POW}_p \circ \mathrm{ROOT}'_q$. Now the growth g of M is in $\Theta(\ell^{p/q})$. The speed-up is controlled

by function f, where $f(\ell) = \ell g(\ell)$. Now f is in $\Theta\left(\ell^{(p+q)/q}\right)$. On the other hand, $f^{-1}(x)$ is in $\Theta\left(x^{q/(p+q)}\right)$.

Consider link machine encoding graphs of size n with a bottom cycle of length $\hat{n}$ such that n is in $\Theta\left(\hat{n} \cdot f^{-1}(T(\hat{n}))\right)$. Here $T\colon \mathbb{N} \to \mathbb{N}$ is the time complexity of a problem Π. Let Π now be the problem of safe 2-colouring, which has time complexity $\Theta(n)$.

Now, we obtain that n is in $\Theta(\hat{n}^{(p+2q)/(p+q)})$ and $\hat{n}$ is in $\Theta\left(n^{(p+q)/(p+2q)}\right)$. It follows from Theorems 26 and 27 that problem Π_M has time complexity $\Theta\left(\hat{n}^{q/(p+q)}\right)$, or equivalently, $\Theta\left(n^{q/(p+2q)}\right)$. Claim (1) follows now by setting $q = r$ and $p = s - 2r$.

The other claims are proved in an analogous manner. □

6. Conclusions

In this dissertation, we studied the connections between distributed computing and mathematical logic, the relationship between time and space complexity as well as the existence of various time complexities in the distributed setting. In Section 6.1, we consider the implications of our work and mention closely related works by other authors. Then, in Section 6.2, we discuss ideas on how the research presented in this dissertation can be continued.

6.1 Significance of the results

The work on logical characterisations has already opened up a new research area, with further research being done on the topic. Also the results on complexity classes are tightly intertwined with current research on distributed complexity theory. In the following, we give a more detailed account on the impact of the different aspects of this dissertation.

6.1.1 Distributed computing and logic

Variants of modal logic were used already in the 1980s in analysing concurrent and distributed systems [12] – modal logic provides a natural way to say, for example, that is is impossible for a system to reach a certain internal state that the system designer wants to avoid. However, our approach is very different: we equate the relational models of modal logic and communication graphs of distributed computing, so that points of the model correspond to nodes of the distributed system instead of possible states of the system. This leads to the natural correspondence between modal formulas and distributed algorithms, having resemblance to descriptive complexity in classical centralised complexity theory.

While our work was the first to take the above-mentioned approach, it was by no means the last. In Publication I, we characterised the *constant-time* variants of a hierarchy of models of distributed computing. This limitation was lifted by Kuusisto [25], who introduced a recursive bisimulation-invariant logic called

modal substitution calculus. This recursive logic made it possible to characterise a certain class of distributed algorithms without the constant-time limitation. Kuusisto [24] provided another example of the application of logic to distributed computing by proving that all universally halting distributed algorithms have necessarily a constant time complexity.

The line of research initiated in Publication I was also continued by Reiter [32, 33]. In the first article [33], he extends the weak models by allowing *alternation* and more complicated acceptance conditions. The resulting class of algorithms turns out to be equal to monadic second-order logic on graphs. The second article [32] considers several *asynchronous* models of distributed computing. He shows these models to be equivalent to a fragment of the modal μ-calculus, and furthermore, makes use of this connection to show that the possibility of losing messages does not affect the computational power of the model. Recently, a model of computation that accommodates unique node identifiers was also characterised by an extension of first-order logic [6].

With the above examples in mind, we claim that mathematical logic has turned out to be a viable and promising approach to study distributed computing.

6.1.2 Constant-space distributed computing

Constant-space models of distributed computing have been studied previously in various forms. One classical example is provided by cellular automata [18, 34, 35]. More recently, networks of finite-state machines have been studied by Emek and Wattenhofer [16] as well as by Kuusisto and Reiter [26, 32, 33]. However, the setting of our work – synchronous communication and deterministic algorithms that halt in a finite but not necessarily constant time – appears to be novel.

Even more importantly, the relationship between space and time is a relatively unexplored topic in distributed computing research. We can ask, for example, what kind of time complexities are possible for problems contained in a given space-limited complexity class – or vice versa. The work presented in Publication III aims to change this and start a new area of research in distributed complexity theory.

On a more concrete level, our result rules out a candidate method for proving upper bounds on time complexity: showing that a problem is solvable in constant space does not necessarily imply that it is solvable in sublinear time. On the other hand, the result does imply that it makes sense to try to come up with a very space-economical algorithm even if the problem in hand has a high time complexity.

In addition to being of theoretical interest, the property of having a constant space complexity has practical consequences. Consider, for instance, wireless sensor networks. An algorithm with constant memory consumption, that is, one that is independent of the size of the network, makes it possible to deploy an arbitrary number of nodes. Another possible application area is provided by nature. It is reasonable to assume that the size of an organism – for example,

a single cell, or an animal that is a part of a swarm or flock – does not depend on the size of the larger ensemble. Therefore, if we can show that a task is not doable in constant space in the distributed setting, we will possibly gain new knowledge on related natural phenomena.

6.1.3 Distributed time complexity classes

The study of complexity classes in the LOCAL model was initiated in the 1990s by Naor and Stockmeyer [30], who introduced the concept of LCL problems. Their work, together with more recent results [9], implied that in the case of cycle and path graphs, the only possible time complexities for LCL problems are $O(1)$, $\Theta(\log^* n)$ and $\Theta(n)$. However, the case of general bounded-degree graphs remained open until very recently: only the three classes $O(1)$, $\Theta(\log^* n)$ and $\Theta(n)$ were known to contain LCL problems, while the existence of other complexities could not be ruled out.

The work of Brandt et al. [7] changed the situation substantially by proving the existence of natural LCL problems with intermediate complexity: Δ-colouring, as well as the so-called sinkless orientation problem, turned out to have a time complexity between $\omega(\log^* n)$ and $o(n)$. A line of research by others followed: for instance, Chang et al. [9] proved that there is a gap between $\omega(\log^* n)$ and $o(\log n)$, and Chang and Pettie [10] showed the existence of problems with complexities $\Theta(n^{1/k})$ for all k. The results of Publication IV are direct continuation to this progress of developing complexity theory for LCL problems. In particular, Chang and Pettie [10] conjectured that there are no problems with complexity between $\omega(n^{1/(k+1)})$ and $o(n^{1/k})$. This was refuted by Publication IV: there exist LCL problems with complexities $\Theta(n^\alpha)$ for any rational number $\alpha < 1/2$.

Knowledge on the existence of different complexity classes can also have practical implications. Gap results often directly give us speed-up theorems: if there is a known gap between time complexity classes $O(f)$ and $\Omega(f')$, and we have an algorithm with time complexity in $o(f')$, we can speed up the algorithm to run in time $O(f)$. On the other hand, hierarchy results such as the one given in Publication IV give useful guidance on where to look for and where to not look for such gaps in time complexity.

6.2 Future research

The work presented in this dissertation provides ample opportunities for future research. In this section, we outline some ideas.

6.2.1 Logical characterisations

The logical characterisations for the weak models that we presented in Publication I, along with the subsequent work mentioned in Section 6.1.1, raise

the question of also finding such characterisations for other models of distributed computation. The standard LOCAL model, for example, is lacking such a characterisation. While it may prove difficult to find a natural logical system that captures the LOCAL model exactly, there exist other models of computation that may be more susceptible to this kind of approach. To give one example, Naor and Stockmeyer [30], and later Göös et al. [19], have studied a so-called *order-invariant* model, where instead of unique identifiers, nodes are linearly ordered. In addition to characterising solvability in a given model of computation, describing classes of problems solvable in certain amount of time or space by means of logic would be a very interesting result.

Further research on this topic has a potential to give new general tools for proving results on distributed computing. In addition to bisimulation, there are tools such as Ehrenfeucht-Fraïssé games [15] that can be used to show that certain property cannot be expressed in a given logic. As was demonstrated in Publication I, it is possible to rephrase such results as impossibility results in distributed computing. Together with more fine-grained logical characterisations, these provide a promising approach for coming up with new lower-bound results. In addition to advancing the research on distributed computing, it is reasonable to expect that further studies on this connection enable the transfer of tools and results from distributed computing to the field of logic.

6.2.2 Computational algorithm design

Formal verification of distributed systems by model checking [11, 22, 23] is a well-established topic, as is the synthesis of distributed protocols based on a specification given by a logical formula [13, 17, 29]. However, the models of computation considered in that research area usually differ from our framework of distributed computing, and more importantly, the connection to logic is different. In the traditional approach, in order to synthesise an algorithm, one needs to construct a model that satisfies a given formula. In the approach of Publication I, synthesising an algorithm corresponds to constructing a formula that satisfies certain properties.

Designing distributed algorithms for models of computation similar to ours by means of computational methods has recently gained momentum [8, 14, 21]. However, the search space of all candidate algorithms is usually huge, which makes the approach feasible only in quite limited cases. A potentially helpful observation here is that fast and correct algorithms are often relatively simple. If the search could be directed towards simple – but otherwise arbitrary – algorithms, the synthesis task might become feasible on a much larger scale. The connection to logic, where each distributed algorithm corresponds to a formula and vice versa, provides a way to formalise this simplicity. This likely requires us to study complexity measures of formulas other than the usual modal depth or quantifier rank. The size of a formula [20] would be a natural candidate.

6.2.3 New problem classes

While there exists an ever increasing amount of results on the complexity of LCL problems in the LOCAL model, there is little knowledge on problems beyond the class of LCL problems. In particular, the case of general graphs – as opposed to bounded-degree graphs – is interesting. It is not clear at all, what is the best way to generalise the definition of LCL problems to general graphs. First, it would be desirable to be able to describe each graph problem in the class in a finite space, which rules out defining a problem by simply listing all the valid neighbourhoods for each possible maximum degree. Second, the definition should lead to an interesting structure of complexity classes – there should be several non-empty classes, with preferably gaps between them.

One approach to generalising the class of LCL problems is as follows. The generalised class would consist of graph problems Π such that each candidate solution S of Π can be verified in a model of computation weaker than the standard LOCAL model. This is where our Publications I and IV come together: we can take the SB (set–broadcast) model (or equivalently, the basic modal logic) from Publication I, modify it slightly to lift the bounded-degree limitation and define the class of SB-*checkable* problems. With finite state and message sets, this immediately gives us a way to define graph problems in a finite way in general graphs. Also, the class of SB-checkable problems contains several classical graph problems such as the maximal independent set and the weak colouring problems. However, it is not clear whether we can develop interesting complexity theory in this setting.

The important thing here is the general approach: instead of trying to generalise the definition of LCL problems directly, characterise a class of problems by means of logic – and then lift the bounded-degree assumption.

6.2.4 New complexity measures

Finally, we can extend the work presented in Publications III and IV by considering complexity measures other than the number of communication rounds or the amount of space required. One such complexity measure would be *volume*: Instead of gathering information from a neighbourhood of a certain radius, nodes would be able to adaptively query nodes whose neighbour they have already seen. Then, volume would be defined as the maximum number of queries conducted by any node.

References

[1] Jean-Paul Allouche and Jeffrey Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In *Proc. International Conference on Sequences and Their Applications (SETA 1998)*, pages 1–16. Springer, 1999. DOI: 10.1007/978-1-4471-0551-0_1.

[2] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 82–93. ACM, 1980. DOI: 10.1145/800141.804655.

[3] Matti Åstrand and Jukka Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010)*, pages 294–302. ACM, 2010. DOI: 10.1145/1810479.1810533.

[4] Patrick Blackburn, Maarten de Rijke and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science 53. Cambridge University Press, 2001. DOI: 10.1017/CBO9781107050884.

[5] Patrick Blackburn, Johan van Benthem and Frank Wolter, editors. *Handbook of Modal Logic*. Studies in Logic and Practical Reasoning 3. Elsevier, 2007.

[6] Benedikt Bollig, Patricia Bouyer and Fabian Reiter. Identifiers in registers – describing network algorithms with logic. In *Proc. 22nd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2019)*. 2019. To appear.

[7] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC 2016)*, pages 479–488. ACM, 2016. DOI: 10.1145/2897518.2897570.

[8] Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela and Przemysław Uznański. LCL problems on grids. In *Proc. 35th ACM Symposium on Principles of Distributed Computing (PODC 2017)*, pages 101–110. ACM, 2017. DOI: 10.1145/3087801.3087833.

[9] Yi-Jun Chang, Tsvi Kopelowitz and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 615–624. IEEE, 2016. DOI: 10.1109/FOCS.2016.72.

[10] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. In *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2017)*, pages 156–167. IEEE, 2017. DOI: 10.1109/FOCS.2017.23.

[11] Bernadette Charron-Bost, Stepahn Merz, Andrey Rybalchenko and Josef Widder. Formal verification of distributed algorithms (Dagstuhl seminar 13141). *Dagstuhl Reports* 3.4 (2013), pages 1–16. DOI: 10.4230/DagRep.3.4.1.

[12] E. M. Clarke, E. A. Emerson and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8.2 (1986), pages 244–263. DOI: 10.1145/5397.5399.

[13] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. 3rd Workshop on Logics of Programs (LOP 1981)*, pages 52–71. Springer, 1982. DOI: 10.1007/BFb0025774.

[14] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela and Siert Wieringa. Synchronous counting and computational algorithm design. *Journal of Computer and System Sciences* 82.2 (2016), pages 310–332. DOI: 10.1016/j.jcss.2015.09.002.

[15] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. 2nd edition. Springer Monographs in Mathematics. Springer, 1995. DOI: 10.1007/3-540-28788-4.

[16] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *Proc. 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC 2013)*, pages 137–146. ACM, 2013. DOI: 10.1145/2484239.2484244.

[17] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 321–330. IEEE, 2005. DOI: 10.1109/LICS.2005.53.

[18] Martin Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American* 223.4 (1970), pages 120–123.

[19] Mika Göös, Juho Hirvonen and Jukka Suomela. Lower bounds for local approximation. *Journal of the ACM* 60.5 (2013), 39:1–23. DOI: 10.1145/2528405.

[20] Lauri Hella and Jouko Väänänen. The size of a formula as a measure of complexity. In *Logic Without Borders: Essays on Set Theory, Model Theory, Philosophical Logic and Philosophy of Mathematics*. Ontos Mathematical Logic 5. De Gruyter, 2015, pages 193–214. DOI: 10.1515/9781614516873.193.

[21] Juho Hirvonen, Joel Rybicki, Stefan Schmid and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *The Electronic Journal of Combinatorics* 24.4 (2017), #P4.21. URL: https://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i4p21.

[22] Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith and Josef Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *Proc. 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2013)*, pages 201–209. IEEE, 2013. DOI: 10.1109/FMCAD.2013.6679411.

[23] Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith and Josef Widder. Towards modeling and model checking fault-tolerant distributed algorithms. In *Proc. 20th International Symposium on Model Checking Software (SPIN 2013)*, pages 209–226. Springer, 2013. DOI: 10.1007/978-3-642-39176-7_14.

[24] Antti Kuusisto. Infinite networks, halting and local algorithms. In *Proc. 5th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 2014)*, pages 147–160. Open Publishing Association, 2014. DOI: 10.4204/EPTCS.161.14.

[25] Antti Kuusisto. Modal logic and distributed message passing automata. In *Proc. 22nd Annual EACSL Conference on Computer Science Logic (CSL 2013)*, pages 452–468. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. DOI: 10.4230/LIPIcs.CSL.2013.452.

[26] Antti Kuusisto and Fabian Reiter. Emptiness problems for distributed automata. In *Proc. 8th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 2017)*, pages 210–222. Open Publishing Association, 2017. DOI: 10.4204/EPTCS.256.15.

[27] Tuomo Lempiäinen. A Classification of Weak Models of Distributed Computing. Master's thesis. Department of Mathematics and Statistics, University of Helsinki, 2014. URL: http://urn.fi/URN:NBN:fi-fe2017112251644.

[28] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing* 21.1 (1992), pages 193–201. DOI: 10.1137/0221015.

[29] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 6.1 (1984), pages 68–93. DOI: 10.1145/357233.357237.

[30] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing* 24.6 (1995), pages 1259–1277. DOI: 10.1137/S0097539793254571.

[31] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Discrete Mathematics and Applications. SIAM, 2000. DOI: 10.1137/1.9780898719772.

[32] Fabian Reiter. Asynchronous distributed automata: a characterization of the modal mu-fragment. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, 100:1–100:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. DOI: 10.4230/LIPIcs.ICALP.2017.100.

[33] Fabian Reiter. Distributed graph automata. In *Proc. 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 192–201. IEEE, 2015. DOI: 10.1109/LICS.2015.27.

[34] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, 1966.

[35] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[36] Masafumi Yamashita and Tsunehiko Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on Parallel and Distributed Systems* 10.9 (1999), pages 878–887. DOI: 10.1109/71.798313.

Aalto-DD 55/2019

ISBN 978-952-60-8477-0 (printed)
ISBN 978-952-60-8478-7 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

DOCTORAL
DISSERTATIONS