$$\delta : Q \times 2^Q \to Q$$

PhD thesis in Theoretical Computer Science

Distributed Automata and Logic

Fabian Reiter

Dissertation defense on 12 December 2017.

Olivier Carton	*supervisor*
Bruno Courcelle	*examiner*
Pierre Fraigniaud	*examiner*
Nicolas Ollinger	*examiner*
Jukka Suomela	*reviewer*
Christine Tasson	*examiner*
Wolfgang Thomas	*reviewer*

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

École Doctorale 386
Sciences Mathématiques de Paris Centre

université PARIS DIDEROT PARIS 7

USPC Université Sorbonne Paris Cité

PhD thesis in Theoretical Computer Science

Distributed Automata and Logic

Fabian Reiter

Paris, 2017

Last revision: 24 November 2017

Contact address: fabian.reiter@gmail.com

Contents

Abstract iii

Résumé v

Acknowledgments vii

1 Introduction 1
- 1.1 Background . . . 2
  - 1.1.1 Related work in automata theory . . . 2
  - 1.1.2 Related work in distributed computing . . . 4
- 1.2 Contributions . . . 7
- 1.3 Outline . . . 9

2 Preliminaries 11
- 2.1 Basic notation . . . 11
- 2.2 Symbols . . . 11
- 2.3 Structures . . . 12
- 2.4 Different kinds of digraphs . . . 13
- 2.5 The considered logics . . . 14
- 2.6 Example formulas . . . 17
- 2.7 Distributed automata . . . 19

3 Alternating Local Automata 21
- 3.1 Informal description . . . 21
- 3.2 Formal definitions . . . 23
- 3.3 Hierarchy and closure properties . . . 28
- 3.4 Equivalence with monadic second-order logic . . . 30
- 3.5 Emptiness problem for nondeterministic automata . . . 32
- 3.6 Summary and discussion . . . 32

4 Asynchronous Nonlocal Automata 37
- 4.1 Preliminaries . . . 37
- 4.2 Equivalence with the backward mu-fragment . . . 41
- 4.3 Computing least fixpoints using asynchronous automata . . . 42
- 4.4 Capturing asynchronous runs using least fixpoints . . . 43

5 Emptiness Problems 49
5.1 Preliminaries . . . 50
5.2 Comparison with classical automata . . . 51
5.3 Exploiting forgetfulness . . . 53
5.4 Exchanging space and time . . . 54
5.5 Timing a firework show . . . 57

6 Alternation Hierarchies 63
6.1 Preliminaries . . . 65
6.2 Separation results . . . 65
6.3 Top-level proofs . . . 67
6.3.1 Figurative inclusions . . . 67
6.3.2 Proving the main theorem . . . 69
6.4 Grids as a starting point . . . 71
6.4.1 The standard translation . . . 71
6.4.2 A detour through tiling systems . . . 72
6.4.3 Equivalent hierarchies on grids . . . 74
6.4.4 A logical characterization of grids . . . 75
6.5 A toolbox of encodings . . . 77
6.5.1 Encodings that allow for translation . . . 77
6.5.2 Getting rid of multiple edge relations . . . 83
6.5.3 Getting rid of vertex labels . . . 85
6.5.4 Getting rid of backward modalities . . . 86
6.5.5 Getting rid of directed edges . . . 87
6.5.6 Getting rid of global modalities . . . 88

7 Perspectives 91
7.1 Focused questions . . . 91
7.1.1 Is there an alternation level that covers first-order logic? . . . 91
7.1.2 Does asynchrony entail quasi-acyclicity? . . . 92
7.1.3 Is asynchrony decidable? . . . 92
7.1.4 Are forgetful automata useful as tree automata? . . . 92
7.1.5 How powerful are quasi-acyclic automata on dipaths? . . . 92
7.2 Broader questions . . . 93
7.2.1 What about distributed automata on infinite digraphs? . . . 93
7.2.2 What is the overlap with cellular automata? . . . 93
7.2.3 Can we characterize more powerful models? . . . 94

Bibliography 95

Index 101

Abstract

Distributed automata are finite-state machines that operate on finite directed graphs. Acting as synchronous distributed algorithms, they use their input graph as a network in which identical processors communicate for a possibly infinite number of synchronous rounds. For the *local* variant of those automata, where the number of rounds is bounded by a constant, Hella et al. (2012, 2015) have established a logical characterization in terms of basic modal logic. In this thesis, we provide similar logical characterizations for two more expressive classes of distributed automata.

The first class extends local automata with a global acceptance condition and the ability to alternate between nondeterministic and parallel computations. We show that it is equivalent to monadic second-order logic on graphs. By restricting transitions to be nondeterministic or deterministic, we also obtain two strictly weaker variants for which the emptiness problem is decidable.

Our second class transfers the standard notion of asynchronous algorithm to the setting of nonlocal distributed automata. The resulting machines are shown to be equivalent to a small fragment of least fixpoint logic, and more specifically, to a restricted variant of the modal μ-calculus that allows least fixpoints but forbids greatest fixpoints. Exploiting the connection with logic, we additionally prove that the expressive power of those asynchronous automata is independent of whether or not messages can be lost.

We then investigate the decidability of the emptiness problem for several classes of nonlocal automata. We show that the problem is undecidable in general, by simulating a Turing machine with a distributed automaton that exchanges the roles of space and time. On the other hand, the problem is found to be decidable in LOGSPACE for a class of forgetful automata, where the nodes see the messages received from their neighbors but cannot remember their own state.

As a minor contribution, we also give new proofs of the strictness of several set quantifier alternation hierarchies that are based on modal logic.

Keywords. Automata, Distributed algorithms, Modal logic, Monadic second-order logic, Graphs.

Résumé

Les automates distribués sont des machines à états finis qui opèrent sur des graphes orientés finis. Fonctionnant en tant qu'algorithmes distribués synchrones, ils utilisent leur graphe d'entrée comme un réseau dans lequel des processeurs identiques communiquent entre eux pendant un certain nombre (éventuellement infini) de rondes synchrones. Pour la variante *locale* de ces automates, où le nombre de rondes est borné par une constante, Hella et al. (2012, 2015) ont établi une caractérisation logique par des formules de la logique modale de base. Dans le cadre de cette thèse, nous présentons des caractérisations logiques similaires pour deux classes d'automates distribués plus expressives.

La première classe étend les automates locaux avec une condition d'acceptation globale et la capacité d'alterner entre des modes de calcul non-déterministe et parallèle. Nous montrons qu'elle est équivalente à la logique monadique du second ordre sur les graphes. En nous restreignant à des transitions non-déterministes ou déterministes, nous obtenons également deux variantes d'automates strictement plus faibles pour lesquelles le problème du vide est décidable.

Notre seconde classe adapte la notion standard d'algorithme asynchrone au cadre des automates distribués non-locaux. Les machines résultantes sont prouvées équivalentes à un petit fragment de la logique de point fixe, et plus précisément, à une variante restreinte du μ-calcul modal qui autorise les plus petits points fixes mais interdit les plus grands points fixes. Profitant du lien avec la logique, nous montrons aussi que la puissance expressive de ces automates asynchrones est indépendante du fait que des messages puissent être perdus ou non.

Nous étudions ensuite la décidabilité du problème du vide pour plusieurs classes d'automates non-locaux. Nous montrons que le problème est indécidable en général, en simulant une machine de Turing par un automate distribué qui échange les rôles de l'espace et du temps. En revanche, le problème s'avère décidable en LOGSPACE pour une classe d'automates oublieux, où les nœuds voient les messages reçus de leurs voisins, mais ne se souviennent pas de leur propre état.

Finalement, à titre de contribution mineure, nous donnons également de nouvelles preuves de séparation pour plusieurs hiérarchies d'alternance de quantificateurs basées sur la logique modale.

Mots-clés. Automates, Algorithmes distribués, Logique modale, Logique monadique du second ordre, Graphes.

Acknowledgments

First and foremost, I would like to thank my advisor, Olivier Carton, for his continuous support during the past three years. This included finding a scholarship for me, spending countless hours with me in front of a whiteboard, as well as helping me in the writing of several papers and this thesis. I am especially grateful for the immense freedom he granted me throughout the entire period, letting me pursue ideas of my own, but at the same time always being available for discussion. He provided guidance whenever I needed it, but never exerted pressure, gave very good advice, but always let me decide for myself. In my opinion, this is exactly how a doctoral thesis should be supervised, but it can by no means be taken for granted. I therefore consider myself very fortunate to have had Olivier as my advisor.

My sincere thanks extend to Bruno Courcelle, Pierre Fraigniaud, Nicolas Ollinger, Jukka Suomela, Christine Tasson, and Wolfgang Thomas, who kindly accepted to be part of my thesis committee.

Jukka Suomela and Wolfgang Thomas did me the great honor of reviewing a preliminary version of the manuscript. They gave detailed and extremely flattering feedback, and both made an important remark that led to a major improvement of this document: I had failed to include a discussion of perspectives for future research. This shortcoming has now been addressed by the addition of Chapter 7. Furthermore, Wolfgang compiled a very helpful collection of suggestions, which I have tried to incorporate into the present version.

Bruno Courcelle graciously gave his time to read my master's thesis in 2014, although he had never heard of me before. He then informed Géraud Sénizergues, who most kindly invited me to the final conference of the FREC project in Marseille. This opened the door for me into the French community of automata theory, especially since I met my future doctoral advisor at that conference. Thus, it is indirectly through Bruno that I came to Paris.

There, at IRIF (formerly LIAFA), Pierre Fraigniaud showed a very kind interest in my work and opened another door for me, this time into the community of distributed computing. He did so by referring to my first paper in several of his own collaborations and by providing various opportunities for me to meet his colleagues, in particular at two international workshops in Bertinoro and Oaxaca. The latter was made possible through a joint effort of Pierre and Sergio Rajsbaum.

In addition to the committee, I am grateful to Fabian Kuhn and Andreas Podelski, who supervised my master's thesis (the starting point for the present thesis), to Antti Kuusisto, who collaborated with me on the work in Chapter 5, to Laurent Feuilloley, who proofread and corrected the overview of distributed decision in Section 1.1.2, to Charles Paperman, who was the first to tell me that I was unknowingly working with some kind of modal logic, to Nicolas Bacquey, who informed me that exchanging space and time is a common technique in cellular automata theory, and to Thomas Colcombet, who developed the knowledge package and encouraged me to use it here.

We have just crossed the dividing line, where I stop mentioning people by name. This may seem hasty, or even harsh, but there are two good reasons. First, I value my privacy very much and do not want to share personal details in a document that will be publicly available on the Internet. Second, the larger the circle of people I include, the greater the risk of forgetting someone. A simple rule that helps to avoid both of these issues is to mention only people who stand in some direct professional relation to the thesis. Nevertheless, many more have helped me over the years and had a tremendous influence on my life and work. I therefore sincerely hope not to offend anyone by expressing my gratitude in the following simplistic manner:

Many thanks to my colleagues, friends, and family!

1

Introduction

The present thesis aims to contribute to the recently initiated development of a *descriptive complexity* theory for *distributed computing*.

What does this mean? Descriptive complexity [Imm99] basically compares the expressive powers of certain classes of *algorithms*, or abstract machines, with those of certain classes of logical *formulas*. The Holy Grail, so to speak, is to establish equivalences of the form:

We identify algorithms with abstract machines.

> "Algorithm class $\mathcal{A}$ has exactly the same power as formula class Φ."

Probably the most famous result in this area is Fagin's theorem from 1974 [Fag74], which roughly states that a graph property can be recognized by a nondeterministic Turing machine in polynomial time if and only if it can be defined by a formula of existential second-order logic. The theorem thereby provides a logical characterization of the complexity class NPTIME.

Distributed computing [Lyn96, Pel00], on the other hand, studies networks composed of several interconnected processors that share a common goal. The processors communicate with each other by passing messages along the links of the network in order to collectively solve some computational problem. In many cases, this is a graph problem, where the considered problem instance is precisely the graph defined by the network itself. All processors run the same algorithm concurrently, and often make no prior assumptions about the size and topology of the graph. Typical problems that can be solved by such *distributed algorithms* include graph coloring, leader election, and the construction of spanning trees and maximal independent sets.

Now, the ultimate objective that motivates this thesis is to develop an extension of descriptive complexity for the classes of algorithms considered in distributed computing. This means that we seek to establish equivalences of the form:

> "Distributed algorithm class $\mathcal{A}$ has the same power as formula class Φ."

However, such a statement can only be substantial if we have a precise definition of class $\mathcal{A}$. Therefore, we will formally represent distributed algorithms as abstract machines, instead of the more common, but informal, representations in pseudocode.

Distributed algorithms are abstract machines that communicate.

Why is this interesting? First and foremost, a descriptive complexity theory for distributed computing would offer the same benefits as its classical counterpart does for sequential computing:

a. If distributed algorithm class $\mathcal{A}$ turns out to be equivalent to formula class Φ, then this provides strong evidence for the naturalness of both classes. Indeed, the definition of any mathematical device may, by itself, seem arbitrary. *Why should distributed machines communicate precisely that way? Why should logical formulas contain precisely those components?* But if two devices, that appear rather different on the surface, turn out to be descriptions of the exact same thing, then this is unlikely to be pure coincidence.

b. Connecting two seemingly unrelated fields – here, distributed computing and logic – can provide new insights into both fields. Some proofs might be easier to perform if one adopts the point of view of one setting rather than the other. Furthermore, some open questions in one field might already have well-known answers in the other. Especially the field of distributed computing could benefit from this, as it is more than a century younger than formal logic, and therefore has had less time to evolve.

Formal logic dates back to the mid-19th century, while distributed computing started in the 1970s/1980s.

Second, distributed computing also brings an interesting new perspective to the field of descriptive complexity itself:

c. Distributed algorithms can be evaluated on the same input as logical formulas, without any need for encoding that input. More precisely, the network in which a distributed algorithm is executed may be considered identical to the structure on which the truth of a corresponding formula is evaluated. This stands in sharp contrast to classical descriptive complexity theory. For instance, in the case of Fagin's theorem, the input of a Turing machine is a binary string that encodes a finite graph in form of an adjacency matrix. Hence, the equivalence of nondeterministic polynomial-time Turing machines and existential second-order logic is actually stated with respect to such an encoding.

1.1 Background

Let us now take a step back and put the subject into context. We start with a brief summary of some classical results in automata theory, and then turn to more recent developments in distributed computing.

1.1.1 Related work in automata theory

Although the field of descriptive complexity theory really started with Fagin's theorem in the 1970s, the idea of characterizing abstract machines through logical formulas had already appeared earlier in automata theory. In the early 1960s, Büchi [Büc60], Elgot [Elg61] and Trakhtenbrot [Tra61] discovered independently of each other that the regular languages, which are recognized by *finite automata* on words, are precisely the languages definable in *monadic second-order logic*, or MSOL (see, e.g., [Tho97b, Thm 3.1]). The latter is an extension of first-order logic, which in addition to allowing quantification over elements of a given domain (e.g., positions in a word), also allows to quantify over sets of such elements. Along with several other equivalent characterizations, in particular through regular expressions [Kle56], regular grammars [Cho56], and finite monoids [Ner58], the equivalence between automata and logic helped to legitimize regularity as a highly natural concept in formal language theory (cf. Item a, above). Furthermore, it proved that the satisfiability and validity problems for MSOL on words are decidable, because so are the corresponding problems for finite automata. In this way, the field of logic directly benefited from the connection with automata theory (cf. Item b). Nowadays, such connections also play a central role in model checking, where one needs to decide whether a system, represented by an automaton, satisfies a given specification, expressed as a logical formula.

About a decade later, the result was generalized from words to labeled trees by Thatcher and Wright [TW68] and Doner [Don70] (see, e.g., [Tho97b, Thm 3.6]). The corresponding tree automata can be seen as a canonical extension of finite automata to trees; as far as MSOL is concerned, the generalization to trees is even more straightforward, since both words and trees are merely special cases of the relational structures on which logical formulas are usually evaluated. The other characterizations of regular languages can also be generalized from words to trees in a natural manner, and quite remarkably, they all remain equivalent on trees (see, e.g., [CDG⁺08]). Hence, the notion of regularity extends directly to tree languages. Moreover, similar equivalences have been established for several other generalizations of words, such as nested words (see [AM09]) and Mazurkiewicz traces (see, e.g., [DM97]).

In contrast, the situation becomes far more complicated if we expand our field of interest to arbitrary finite graphs (possibly with node labels and multiple edge relations). Although some of the characterizations mentioned above can be generalized to graphs in a meaningful way, they are, in general, no longer equivalent. The logical approach is certainly the easiest to generalize, since graphs are yet another special case of relational structures. While on words and trees the existential fragment of MSOL (EMSOL) is already sufficient to characterize regularity, it is strictly less expressive than full MSOL on graphs, as has been shown by Fagin in [Fag75]. Similarly, the algebraic approach (based on monoids) has been adapted to graphs by Courcelle in [Cou90], and it turns out that MSOL is strictly less powerful than his notion of recognizability. (The latter is defined in terms of homomorphisms into many-sorted algebras that are finite in each sort.) A common pattern that emerges from such results is that the different characterizations of regularity drift apart as the complexity of the considered structures increases. In this sense, regularity cannot be considered a natural – or even well-defined – property of graph languages.

Fagin's result was later extended by Matz, Schweikardt and Thomas to yield a complete separation of the MSOL quantifier alternation hierarchy (see [MST02]).

To complicate matters even further, the automata-theoretic characterization which is instrumental in the theory of word and tree languages, does not seem to have a natural counterpart on graphs. A word or tree automaton can scan its entire input in a single canonical traversal, which is completely determined by the structure of the input (i.e., left-to-right, for words, and bottom-up, for trees). On arbitrary graphs, however, there is no sense of a global direction that the automaton could follow, especially since we do not even require connectivity or acyclicity. This is one of the reasons why much research in the area of graph languages has focused on MSOL. In the words of Courcelle and Engelfriet [CE12, p. 3]:

> *… monadic second-order logic can be viewed as playing the role of "finite automata on graphs" …*

Another approach, investigated by Thomas in [Tho91] and [Tho97a], is to nondeterministically assign a state of the automaton to each node of the graph, and then check that this assignment satisfies certain local "transition" conditions for each node (specified with respect to neighboring nodes within a fixed radius) as well as certain global occurrence conditions at the level of the entire graph. The *graph acceptors* devised by Thomas turn out to be equivalent to EMSOL on graphs of bounded degree. Following up on this idea in [SB99], Schwentick and Barthelmann have also suggested a more general model, which remains very close to a normal form of EMSOL, but overcomes the constraint of boundedness on the degree. Both of these graph automaton models are legitimate generalizations of classical finite automata, in the sense that they are equivalent to them and can easily simulate them if we restrict the input to (graphs representing) words or trees. However, on arbitrary graphs, they are less well-behaved, which is a direct consequence of their equivalence with EMSOL. In particular, they do not satisfy closure under complementation, and their emptiness problem is undecidable. It is worth noting that both models are somewhat similar to the local distributed algorithms considered in the next section, insofar as they take into account the local view that each node has of its fixed-radius neighborhood. This connection has already been recognized and exploited by Göös and Suomela in [GS11, GS16]; we will mention it again below.

1.1.2 Related work in distributed computing

Rather surprisingly, the idea of extending descriptive complexity theory to the setting of distributed computing seems to be relatively new. The first research in that direction (of which the author is aware) started in the early 2010s as a collaboration between the Finnish communities of logic and distributed algorithms.

In [HJK+12, HJK+15], Hella et al. have presented a systematic study of several models of distributed computing that impose restrictions of varying degrees on the communication between the nodes of a network. Their most permissive model corresponds to the well-established port-numbering model, where every node has a separate communication channel with each of its neighbors and is guaranteed that the messages sent and received through that channel relate consistently to the same neighbor; the network is anonymous in the sense that nodes are not equipped with unique identifiers. In the nomenclature of [HJK+12, HJK+15], the class of graph problems solvable in this model by deterministic *synchronous* algorithms is denoted by $\mathrm{VV_c}$. Here, "synchronous" means that all nodes of the network share a global clock, thereby allowing the computation to proceed in an infinite sequence of rounds; in each round, all the nodes simultaneously exchange messages with their neighbors, and then update their local state based on the newly obtained information. Next, by dropping the channel-consistency guarantee, one obtains the class VV, where in each round, every node sees a vector consisting of all the incoming messages received from its neighbors, and generates a vector of outgoing messages that are sent to the neighbors; the difference with $\mathrm{VV_c}$ is that the two vectors are not necessarily sorted in the same order, so the node cannot assume that the neighbor who sends the i-th incoming message is the same who receives the i-th outgoing message. (However, the sorting orders do not change throughout the rounds.) Communication is further restricted in the classes MV and SV, where the vector of incoming messages is replaced by a multiset and a set, respectively. In the former case, a node cannot identify the senders of its incoming messages, whereas in the latter, it cannot even distinguish between several identical messages. Similarly, the classes VB, MB, and SB are characterized by the fact that the outgoing vector is replaced by a singleton, meaning that a node must broadcast the same message to all of its neighbors.

The classes of Hella et al.:

	Incoming	*Outgoing*
$\mathrm{VV_c}$	vector	vector
VV	——	——
MV	multiset	——
SV	set	——
VB	vector	singleton
MB	multiset	——
SB	set	——

The main result of [HJK+12, HJK+15] is that the preceding classes satisfy the linear order

$$\text{SB} \subsetneq \text{MB} = \text{VB} \subsetneq \text{SV} = \text{MV} = \text{VV} \subsetneq \text{VV}_{\text{C}}.$$

The same order holds for the so-called *local* (or constant-time) versions of these classes, which contain only those graph problems that can be solved in a constant number of communication rounds, regardless of the size of the network. (For a relatively recent survey of local algorithms, see [Suo13].)

Most relevant for the present thesis, the same paper also establishes a very natural correspondence between these local classes and several variants of modal logic. In particular, a graph property lies in SB(1), the local version of SB, if and only if it can be defined by a formula of *backward modal logic*. Just like a distributed algorithm, such a formula is evaluated from the local point of view of a particular node in the input graph. In order to make a statement about the incoming neighborhood of that node, backward modal logic allows to move the current point of evaluation to one of the incoming neighbors by means of a special operator, called *backward modality*. The key insight of Hella et al. is that the nesting depth of these modalities corresponds precisely to the running time of the local algorithms that solve problems in SB(1). With this idea in mind, it is possible to derive similar characterizations for the other local classes MB(1), …, $\text{VV}_{\text{C}}(1)$ in terms of extensions of backward modal logic that offer additional types of modalities (viz., multimodal and graded modal logic).

Using the backward version of standard modal logic is merely a presentational choice, motivated by the intuition that the messages of a distributed algorithm should flow in the same direction as the network links on which they travel. The presentation in [HJK+12, HJK+15] *is a bit different.*

Motivated by these results, the connection between distributed algorithms and modal logic was further investigated by Kuusisto in [Kuu13a] and [Kuu14]. The first paper lifts the constraint of locality required in [HJK+12, HJK+15], thereby allowing algorithms with arbitrary running times. Now, for local algorithms, it does not matter whether we impose a restriction on the amount of memory space used by each node, because in a constant number of rounds, a node can only visit a constant number of different states. Therefore the local algorithms characterized by Hella et al. are implicitly finite-state machines. On the other hand, in the nonlocal case considered by Kuusisto, space restrictions have to be made explicit. His papers focus on algorithms for the class SB, since results for that class can easily be adapted to the others. In [Kuu13a], particular attention is devoted to a category of such algorithms that act as finite-state semi-deciders; we shall refer to them as *distributed automata*. The main result establishes a logical characterization of distributed automata in terms of a new recursive logic dubbed *modal substitution calculus*. In the same vein, it is also shown that the infinite-state generalizations of distributed automata recognize precisely those graph properties whose complement is definable by the conjunction of a possibly infinite number of backward modal formulas (called modal theory). Furthermore, it is proven that on finite graphs, distributed automata are strictly more expressive than the least-fixpoint fragment of the backward μ-calculus. This logic, which we shall refer to simply as the *backward μ-fragment*, extends backward modal logic with a least fixpoint operator that may not be negated. It thus allows to express statements using least fixpoints, but unlike in the full backward μ-calculus, greatest fixpoints are forbidden. Finally, the second paper [Kuu14] makes crucial use of the connection with logic to show that universally halting distributed automata are necessarily local if infinite graphs are allowed into the picture.

Closely related to the work mentioned above, the last decade has also seen active research in *distributed decision* [FF16], a field that aims to develop a counterpart of computational complexity theory for distributed computing. In that context, the nodes of a given network have to collectively decide whether or not their network satisfies some global property. Every node first computes a local answer, based on the information received from its neighbors over several rounds of communication, and then all answers are aggregated to produce a global verdict. Typically, the network is considered to be in a valid state if it has been unanimously accepted by all nodes; in other words, the global answer is the logical conjunction of the local answers.

Just as in classical complexity theory, a common approach in distributed decision is to start with some base class of deterministic algorithms, and then extend it with additional features, such as nondeterminism and randomness. However, depending on the underlying model of distributed computing, these additional features can quickly lead to excessive expressive power. For instance, if we add unrestricted nondeterminism to the widely adopted LOCAL model, then the nodes can simply guess a representation of the entire network and verify in one round that their guess was correct. Consequently, nondeterministic algorithms in the LOCAL model can already decide every Turing-decidable graph property in a single round of communication (see, e.g., [FF16, § 4.1.1]). To make things more interesting, one therefore often imposes a restriction on the number of bits that each node can nondeterministically choose; viewing nondeterminism as the ability to "guess and verify", we refer to the bit strings guessed by the nodes as *certificates*. A typically chosen bound on the size of those certificates is logarithmic in the size of the network because this allows each node to guess only a constant number of processor identifiers. In stark contrast to the unbounded case, where Turing-decidability is the only limit, there are natural decision problems that cannot be solved by any nondeterministic local algorithm whose certificates are logarithmically bounded. An example of such a problem is to verify whether a given tree is a minimum spanning tree, as has been shown by Korman and Kutten in [KK07]. Nevertheless, on connected graphs, nondeterminism with logarithmic certificates provides enough power to decide every property definable in EMSOL within a constant number of rounds, essentially by using nondeterministic bits to construct a spanning tree and simulate existential set quantifiers. This observation has been made by Göös and Suomela in [GS11, GS16], based on the work of Schwentick and Barthelmann mentioned in the previous section.

The LOCAL model allows unbounded synchronous communication between Turing-complete processors that are equipped with unique identifiers. Despite the name, algorithms in this model are not necessarily local.

Once existential quantification has been introduced into the system, a natural follow-up is to complement it with universal quantification; for instance, in classical complexity theory, alternating the two types of quantifiers leads to the polynomial hierarchy, which generalizes the classes NPTIME and co-NPTIME. While not very interesting for the unrestricted LOCAL model with unbounded certificates (where nondeterminism already suffices to decide everything possible), this form of alternation provides a genuine increase of power if we consider distributed algorithms that are oblivious to the node identifiers. In [BDFO17], Balliu, D'Angelo, Fraigniaud and Olivetti showed that we require one alternation between universal and existential quantifiers in order to be able to decide every Turing-decidable property in the identifier-oblivious variant of the LOCAL model (with unbounded certificates); hence the corresponding alternation hierarchy collapses to its second level. On the other hand, the hierarchy of the standard LOCAL model with certificates of logarithmic size is much less well understood; in particular, it is still open whether or not that hierarchy is infinite. As a first step towards an answer, Feuilloley, Fraigniaud and Hirvonen showed in [FFH16] that if there is equality between the existential and universal versions of a given level in the logarithmic hierarchy, then the entire hierarchy collapses to that level. Furthermore, they could identify a decision problem that lies outside of the hierarchy, which shows that even with the full power of alternation, algorithms whose certificates are logarithmically bounded remain weaker than their unrestricted counterparts.

1.2 Contributions

Obviously, developing a descriptive complexity theory for distributed computing is a highly ambitious project, of which the present work can only strive to be a small building block. As its title suggests, this thesis does not deal with the powerful models of computation that are usually considered in distributed computing. Instead, it takes an automata-theoretic approach and focuses on a rather weak model that has already been explored by Hella et al. and Kuusisto, namely distributed automata. The main contributions are two new logical characterizations related to that model.

The first covers a variant of local distributed automata, extended with a global acceptance condition and the ability to alternate between nondeterministic decisions of the individual processors and the creation of parallel computation branches. This kind of alternation constitutes a canonical generalization of nondeterminism, and is nowadays standard in automata theory. We show that the resulting *alternating local automata with global acceptance* are equivalent to MSOL on finite directed graphs. In spirit, they are similar to the alternation hierarchies considered in the distributed-decision community, even though their expressive power is much more restricted. They also share some similarities with Thomas' graph acceptors, as they use a combination of local conditions, checked by the nodes based on their neighborhood, and global conditions, checked at the level of the entire graph. However, both types of conditions are much simpler than in Thomas' model, which allows us to consider graphs of unbounded degree. To a certain extent, the equivalence with MSOL can be considered as a generalization to graphs of the classical result of Büchi, Elgot and Trakhtenbrot, although the machines involved are by no means deterministic; whereas on words and trees, alternation simply provides a more succinct representation of deterministic automata, it turns out to be a crucial ingredient in our case. If we allow only nondeterminism, we get a model that is not closed under complementation, and is even strictly weaker than EMSOL, but has a decidable emptiness problem. Interestingly, that model is still powerful enough to characterize precisely the regular languages when restricted to words or trees. Hence, this work also contributes to the general observation, made in Section 1.1.1, that regularity becomes a moving target when lifted to the setting of graphs.

The second main contribution consists in a logical characterization of a fully deterministic class of nonlocal automata. As mentioned in Section 1.1.2, Kuusisto has noticed that distributed automata, in their unrestricted form, are strictly more powerful than the backward μ-fragment on finite graphs. While it is straightforward to evaluate any formula of the backward μ-fragment via a distributed automaton, there also exist automata that exploit the fact that a node can determine if it receives the same information from all of its neighbors at the exact same time. Such a behavior cannot be simulated in the backward μ-fragment, and actually not even in the much more expressive MSOL. However, since the argument relies solely on synchrony, it seems natural to ask whether removing this feature can lead to a distributed automaton model that has the same expressive power as the backward μ-fragment. To answer this question, we introduce several classes of *asynchronous automata* that transfer the standard notion of asynchronous algorithm to the setting of finite-state machines. Basically, this means that we eliminate the global clock from the network, thus making it possible for nodes to operate at different speeds and for messages to be delayed for arbitrary amounts of time, or even be lost. From the syntactic point of view, an asynchronous automaton is the same as a synchronous one, but it has to satisfy an additional semantic condition: its acceptance behavior must be independent of any timing-related issues. Taking a closer look at the automata obtained by translating formulas of the backward μ-fragment, we can easily see that they are in fact asynchronous. Furthermore, their state diagrams are almost acyclic, except that the states are allowed to have self-loops; we call this property *quasi-acyclic*. As it turns out, the two properties put together are sufficient to give us the desired characterization: quasi-acyclic asynchronous automata are equivalent to the backward μ-fragment on finite graphs. Incidentally, this remains true even if we consider a seemingly more powerful variant of asynchronous automata, where all messages are guaranteed to be delivered.

Another aspect of distributed automata investigated in this thesis are decision problems, and more specifically emptiness problems, where the task is to decide whether a given automaton accepts on at least one input graph. As all the equivalences mentioned above are effective, we can immediately settle the decidability of the emptiness problem for local automata: it is decidable for the basic variant of Hella et al., but undecidable for the alternating extension that we shall consider. This is because the (finite) satisfiability problem is PSPACE-complete for (backward) modal logic but undecidable for MSOL. The problem is also decidable for our classes of asynchronous automata, since (finite) satisfiability for the (backward) μ-calculus is EXPTIME-complete. However, the corresponding question for unrestricted, nonlocal automata was left open in [Kuu13a]. Here, we answer this question negatively for the general case and also consider it for three special cases. On the positive side, we obtain a LOGSPACE decision procedure for a class of *forgetful* automata, where the nodes see the messages received from their neighbors but cannot remember their own state. When restricted to the appropriate families of graphs, these forgetful automata are equivalent to classical finite word automata, but strictly more expressive than finite tree automata. On the negative side, we show that the emptiness problem is already undecidable for two heavily restricted classes of distributed automata: the quasi-acyclic ones, and those that reject immediately if they receive more than one distinct message per round. For the latter class, we present a proof with an unusual twist, where a Turing machine is simulated by a distributed automaton in such a way that the roles of space and time are reversed between the two devices.

Finally, as a minor contribution, we investigate the problem of separating quantifier alternation hierarchies for several classes of formulas that are based on modal logic. Essentially, these classes are hybrids, obtained by adding the set quantifiers of MSOL to some variant of modal logic. They are motivated by the above characterizations of local distributed automata in terms of (backward) modal logic and MSOL. The contribution is a toolbox of simple encoding techniques that allow to easily transfer to the modal setting the separation results for MSOL established by Matz, Schweikardt and Thomas in [MT97, Sch97, MST02]. We thereby provide alternative proofs to similar findings previously reported by Kuusisto in [Kuu08, Kuu15].

1.3 Outline

The structure of this thesis is rather straightforward. All the notions that occur in several places are defined in Chapter 2. In particular, there is a simple definition of distributed automata that subsumes most of the variants we shall consider. The subsequent four chapters (i.e., 3 to 6) are independent of each other and thus can be read in any order. In Chapter 3, we focus on local distributed automata and present the alternating variant with global acceptance, which is shown to be equivalent to MSOL. Chapter 4 shifts the focus to nonlocal automata; there we introduce the semantic notion of asynchrony and show that quasi-acyclic asynchronous automata are captured by the backward μ-fragment. Nonlocal automata are also the subject of Chapter 5, where we present both positive and negative decidability results on the emptiness problem for several restricted classes. Then, in Chapter 6, we switch completely to logic and consider issues related to quantifier alternation hierarchies. Finally, some perspectives for future research are briefly outlined in Chapter 7.

Note to the reader of the electronic version. The PDF version of this document makes extensive use of hyperlinks. In addition to the cross-reference links inserted automatically by the standard LaTeX package hyperref, most of the notions defined within the document are linked to their point of definition. This new feature, which concerns both text and mathematical notation, is based on the knowledge package developed by Thomas Colcombet. Beware that there can be several links within a single symbolic expression; for instance, the expression $[\![\mathrm{BC}\,\Sigma_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}}$ contains links to five different concepts: $[\![\ldots]\!]$, BC, $\Sigma_\ell^{\mathrm{MSO}}$, $\overrightarrow{\mathrm{ML}}$, and $@\mathrm{DG}$.

2
Preliminaries

This chapter introduces essential notation and terminology that will be recurring throughout this thesis. It is meant to be consulted for specific information rather than for consecutive reading. Concepts that are specific to a single chapter, will be introduced later, along with the topic.

2.1 Basic notation

We denote the empty set by $\varnothing$, the set of Boolean values by $\mathbb{2} = \{0, 1\}$, the set of non-negative integers by $\mathbb{N} = \{0, 1, 2, \dots\}$, the set of positive integers by $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$, and the set of integers by $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$.

Integer intervals of the form $\{i \in \mathbb{Z} \mid m \leqslant i \leqslant n\}$, where $m, n \in \mathbb{Z}$ and $m \leqslant n$, will sometimes be denoted by $[m : n]$. We may also use the shorthand $[n] \coloneqq [1 : n]$, and, by analogy with the Bourbaki notation for real intervals, we indicate that we exclude an endpoint by reversing the square bracket corresponding to that endpoint, e.g., $]m : n] \coloneqq [m : n] \setminus \{m\}$.

For any two sets S and T, the set of all functions from S to T is denoted T^S. This notation gives rise to two important special cases. First, we write $\mathbb{2}^S$ for the power set of S, since we can identify it with the set of all functions from S to $\{0, 1\}$. Second, given $k \in \mathbb{N}$, we write $S^k \coloneqq S^{[k]}$ for the set of all k-tuples over S, since we can identify it with the set of functions from $[k]$ to S. All of these notations have another special case in common: the set of binary strings of length k, denoted $\mathbb{2}^k$, can be interpreted as either the function space from $[k]$ to $\mathbb{2}$, or the power set of $[k]$, or the set of k-tuples over $\mathbb{2}$. By the first interpretation, the individual letters of a string x of length k will be denoted $x(1), \dots, x(k)$. Furthermore, we write $|S|$ for the cardinality of S and $|x|$ for the length of x.

2.2 Symbols

Since logic plays an important role in this thesis, it also has an influence on how we present other concepts; in particular, our definition of directed graphs in Section 2.4

will refer to the notion of (abstract) symbol.

We shall not always make a sharp distinction between variables and (non-logical) constants. Instead, there is simply a fixed supply of symbols, which can serve both as variables and as constants. Hence, the terms "*variable*" and "*constant*" are just synonyms for "symbol"; we will use them whenever we want to clarify the intended role of a symbol within a given context.

The set $\mathbb{S}_0$ contains our *node symbols*, which within formulas will represent nodes of structures such as graphs; among them, there is a special *position symbol* @. Moreover, for every integer $k \geqslant 1$, we let $\mathbb{S}_k$ denote the set of k*-ary relation symbols*. All of these sets are infinite and pairwise disjoint. If a symbol lies in $\mathbb{S}_k$, for $k \geqslant 0$, then we call k the *arity* of that symbol. We also denote the set of all *symbols* by $\mathbb{S}$, i.e., $\mathbb{S} \coloneqq \bigcup_{k \geqslant 0} \mathbb{S}_k$, and shall often refer to the unary relation symbols in $\mathbb{S}_1$ as *set symbols*.

$\mathbb{S}$ contains both variables and constants.

Node symbols will always be represented by lower-case letters, and relation symbols by upper-case ones, often decorated with subscripts. Typically, we use x, y, z for node variables or arbitrary node symbols, X, Y, Z for set variables or arbitrary set symbols, P, Q for set constants, and R, S for relation constants of higher arity or arbitrary symbols. (See Section 2.6 for some simple examples.)

2.3 Structures

Before we formally introduce directed graphs in the next section, we define the more general concept of a relational structure. Although the present thesis focuses mainly on variants of directed graphs, this top-down approach will allow us to specify the semantics of several types of logical formulas in a unified framework, using a consistent notation. In particular, it will be apparent that modal logic simply provides an alternative syntax for a certain fragment of first-order logic (see Section 2.5).

Let σ be any subset of $\mathbb{S}$. A (relational) *structure* G of *signature* σ consists of a nonempty set of *nodes* V^G (also called the *domain* of G), a node x^G of V^G for each node symbol x in σ, and a k-ary relation R^G on V^G for each k-ary relation symbol R in σ. Here, x^G and R^G are called G's *interpretations* of the symbols x and R. We may also say that G is a structure *over* σ, or that σ is the *underlying signature* of G, and we denote σ by $\mathrm{sig}(G)$. In case the position symbol @ lies in $\mathrm{sig}(G)$, we call G a *pointed structure* and $@^G$ the *distinguished node* of G.

A set of structures will be referred to as a *structure language*. As is customary, we are only interested in structures up to isomorphism. That is, two structures over σ are considered to be equal if there is a bijection between their domains that preserves the interpretations of all symbols in σ. Consequently, our structure languages characterize only properties that are invariant under isomorphism.

Let G be a structure and α be a map of the form $\{S_1 \mapsto I_1, \ldots, S_n \mapsto I_n\}$ that assigns to each symbol $S_i \in \mathbb{S}$, for $i \in [n]$, a suitable interpretation I_i over the domain of G. That is, if $S_i \in \mathbb{S}_0$, then $I_i \in V^G$, and if $S_i \in \mathbb{S}_k$, for $k \geqslant 1$, then $I_i \subseteq (V^G)^k$. We use the notation $G[\alpha]$ to designate the α*-extended variant* of G, which is the structure G' obtained from G by interpreting each symbol S_i as I_i, while maintaining the other interpretations provided by G. More formally, letting $\sigma = \{S_1, \ldots, S_n\}$, we have $V^{G'} = V^G$, $\mathrm{sig}(G') = \mathrm{sig}(G) \cup \sigma$, $S_i^{G'} = I_i$ for $i \in [n]$, and $T^{G'} = T^G$ for $T \in \mathrm{sig}(G) \setminus \sigma$. Often, we do not want to give an explicit name to the assignment α, in which case we may denote G' by $G[S_1, \ldots, S_n \mapsto I_1, \ldots, I_n]$. If the interpretations of the symbols in σ are clear from context, we may also refer to G' as the σ*-extended* *variant* of G. Furthermore, as we will often consider pointed variants of structures, we introduce the shorthand $G[v] := G[@ \mapsto v]$ for $v \in V^G$, and refer to $G[v]$ as the *v-pointed variant* of G (i.e., the variant of G with distinguished node v).

2.4 Different kinds of digraphs

The structures we are actually interested in are several variants of directed graphs; these are structures with finite domains and relations of arity at most 2. To facilitate lookup and comparison, we present them all in the same section. In the following definitions, let s and r be non-negative integers.

An *s-bit labeled, r-relational directed graph*, abbreviated *digraph*, is a finite structure G of signature $\{P_1, \dots, P_s, R_1, \dots, R_r\}$, where $P_1, \dots, P_s$ are set symbols, and $R_1, \dots, R_r$ are binary relation symbols.

The sets $P_1^G, \dots, P_s^G$, which we shall call *labeling sets*, determine a (node) *labeling* $\lambda^G : V^G \to 2^s$ that assigns a binary string of length s to each node. More precisely, we define λ^G such that

$$\lambda^G(v)(i) = \begin{cases} 0 & \text{if } v \notin P_i, \\ 1 & \text{otherwise,} \end{cases}$$

for all $v \in V^G$ and $i \in [s]$. Given another mapping $\zeta : V^G \to 2^{s'}$ with $s' \in \mathbb{N}$, we shall denote by $G[\zeta]$ the *ζ-relabeled variant* of G, i.e., the s'-bit labeled digraph G' that is the same as G, except that its labeling $\lambda^{G'}$ is equal to ζ.

Our bracket notation is overloaded, but if one knows the type of ζ, the ζ-relabeled variant $G[\zeta]$ of G should be easy to distinguish from an α-extended variant $G[\alpha]$, as well as from a v-pointed variant $G[v]$.

It is often convenient to regard the labels of an s-bit labeled digraph as the binary encodings of letters of some finite alphabet Σ. With respect to a given injective map $f : \Sigma \to 2^s$, a *Σ-labeled digraph* is an s-bit labeled digraph G such that for every node $v \in V^G$, we have $\lambda^G(v) = f(a)$ for some $a \in \Sigma$. Since we do not care about the specific encoding function f, we will never mention it explicitly, and just call G a Σ-labeled, r-relational digraph.

The binary relations $R_1^G, \dots, R_r^G$ will be referred to as *edge relations*. If uv is an *edge* in R_i^G, then u is called an *incoming i-neighbor* of v, or simply an incoming neighbor, and v is called an *outgoing i-neighbor* of u, or just outgoing neighbor. We also say that u and v are *adjacent*, and without further qualification, the term *neighbor* refers to both incoming and outgoing neighbors. The (undirected) *neighborhood* of a node is the set of all of its neighbors, and the *incoming* and *outgoing* neighborhoods are defined analogously. A node without incoming neighbors is called a *source*, whereas a node without outgoing neighbors is called a *sink*.

The class of all s-bit labeled, r-relational digraphs is denoted by DG_s^r. In case the parameters are $s = 0$ and $r = 1$, we may omit them and use the shorthand $\mathrm{DG} := \mathrm{DG}_0^1$. We shall also drop the subscripts on the symbols, and just write P or R, if there is only one symbol of a given arity. Furthermore, we denote by DG_Σ^r the class of all Σ-labeled, r-relational digraphs.

As can be easily guessed from the previous definitions, a *pointed digraph* is a digraph in which some node has been marked by the position symbol @, i.e., it is a structure of the form $G[@ \mapsto v]$, with $G \in \mathrm{DG}_s^r$ and $v \in V^G$. We write $@\mathrm{DG}_s^r$ for the set of all s-bit labeled, r-relational pointed digraphs, and define $@\mathrm{DG} := @\mathrm{DG}_0^1$.

A digraph G is called an (s-bit labeled, r-relational) *undirected graph*, or simply *graph*, if all of its edge relations are irreflexive and symmetric, i.e., if for all $u, v \in V^G$ and $i \in [r]$, it holds that $uu \notin R_i^G$, and $uv \in R_i^G$ if and only if $vu \in R_i^G$. The corresponding class is denoted by $\textsc{graph}^r_s$, and we may use the shorthand $\textsc{graph} := \textsc{graph}^1_0$.

A digraph G is (weakly) *connected* if for every nonempty proper subset W of V^G, there exist two nodes $u \in W$ and $v \in V^G \setminus W$ that are adjacent.

The node labeling λ^G of a Σ-labeled digraph constitutes a valid *coloring* of G if no two adjacent nodes share the same label, i.e., if $uv \in R^G_i$ implies $\lambda^G(u) \neq \lambda^G(v)$, for all $u, v \in V^G$ and $i \in [r]$. If $|\Sigma| = k$, such a coloring is called a k-coloring of G, and any r-relational digraph for which a k-coloring exists is said to be *k-colorable*. Note that, by definition, a digraph that contains self-loops is not k-colorable for any k.

A *directed rooted tree*, or *ditree*, is an (s-bit labeled) r-relational digraph G that has a distinct node v_ϵ, called the *root*, such that $R^G_i \cap R^G_j = \varnothing$ for $i \neq j$, and from each node v in V^G, there is exactly one way to reach v_ϵ by following the directed edges in $\bigcup_{1 \leqslant i \leqslant r} R^G_i$. A *pointed ditree* is a pointed digraph $G[v_\epsilon]$, where G is a ditree and v_ϵ is its root. Moreover, a (pointed) r-relational ditree is called *ordered* if for $1 \leqslant i \leqslant r$, every node has at most one incoming i-neighbor and every node that has an incoming $(i+1)$-neighbor also has an incoming i-neighbor. As a special case, an ordered 1-relational ditree is referred to as a *directed path*, or *dipath*. Accordingly, the distinguished node of a *pointed dipath* is the last node (the one with no outgoing neighbor). The classes of pointed dipaths and pointed ordered ditrees can be identified with the structures on which classical word and tree automata are run. We denote them by $@\textsc{dipath}_s$ and $@\textsc{oditree}^r_s$, respectively.

We shall also consider an important subclass of $\textsc{dg}^2_s$ whose members represent rectangular labeled grids (also called pictures). In such a structure G, each node is identified with a grid cell, and the edge relations R^G_1 and R^G_2 are interpreted as the "vertical" and "horizontal" successor relations, respectively. The unique node that has no predecessor at all is regarded as the "upper-left corner", and all the usual terminology of matrices applies. Formally, G is a *s-bit labeled grid* if, for some $m, n \geqslant 1$, it is isomorphic to a structure with domain $\{1, \dots, m\} \times \{1, \dots, n\}$ and edge relations

$$R^G_1 = \Big\{ \big((i,j), (i+1, j)\big) \;\Big|\; 1 \leqslant i < m,\ 1 \leqslant j \leqslant n \Big\},$$
$$R^G_2 = \Big\{ \big((i,j), (i, j+1)\big) \;\Big|\; 1 \leqslant i \leqslant m,\ 1 \leqslant j < n \Big\}.$$

If $s = 0$, we refer to G simply as a *grid*. In alignment with the previous nomenclature, we let $\textsc{grid}$ and $\textsc{grid}_s$ denote the classes of grids and s-bit labeled grids.

A *digraph language* is a structure language that consist of digraphs with a fixed number of labeling sets and edge relations, i.e., a subset of $\textsc{dg}^r_s$, for some $s, r \in \mathbb{N}_+$. The notion is defined analogously for all the other classes of structures introduced above. In particular, a *pointed-digraph language* is a subset of $@\textsc{dg}^r_s$.

2.5 The considered logics

As we shall contemplate both classical logic and several variants of modal logic, we introduce them all in a common framework. First we define the syntax and semantics of a generalized language, and then we specify which particular syntactic fragments we are interested in. Some examples will follow in Section 2.6.

Table 2.1 shows how *formulas* are built up, and what they mean. Furthermore, it indicates how to obtain the set $\mathrm{free}(\varphi)$ of symbols that occur *freely* in a given

Syntax Formula ψ	*Free symbols* Symbol set $\mathrm{free}(\psi)$	*Semantics* Necessary and sufficient condition for $G \vDash \psi$
x	$\{@, x\}$	$@^G = x^G$
$(x \doteq y)$	$\{x, y\}$	$x^G = y^G$
X	$\{@, X\}$	$@^G \in X^G$
$X(x)$	$\{x, X\}$	$x^G \in X^G$
$R(x_0, \ldots, x_k)$	$\{x_0, \ldots, x_k, R\}$	$(x_0^G, \ldots, x_k^G) \in R^G$
$\neg\varphi$	$\mathrm{free}(\varphi)$	not $G \vDash \varphi$
$(\varphi_1 \vee \varphi_2)$	$\mathrm{free}(\varphi_1) \cup \mathrm{free}(\varphi_2)$	$G \vDash \varphi_1$ or $G \vDash \varphi_2$
$\langle R \rangle(\varphi_1, \ldots, \varphi_k)$	$\{@, R\} \cup \bigcup_{1 \leqslant i \leqslant k} \mathrm{free}(\varphi_i)$	For some $v_1, \ldots, v_k \in V^G$ such that $(@^G, v_1, \ldots, v_k) \in R^G$, we have $G[@ \mapsto v_i] \vDash \varphi_i$ for each $i \in \{1, \ldots, k\}$.
$\overleftarrow{\langle R \rangle}(\varphi_1, \ldots, \varphi_k)$	same as above	As above, except for the condition $(v_k, \ldots, v_1, @^G) \in R^G$.
$\langle\cdot\rangle\varphi$	$\mathrm{free}(\varphi) \setminus \{@\}$	$G[@ \mapsto v] \vDash \varphi$ for some $v \in V^G$
$\exists_x \varphi$	$\mathrm{free}(\varphi) \setminus \{x\}$	$G[x \mapsto v] \vDash \varphi$ for some $v \in V^G$
$\exists_X \varphi$	$\mathrm{free}(\varphi) \setminus \{X\}$	$G[X \mapsto W] \vDash \varphi$ for some $W \subseteq V^G$
Here, $x, x_0, \ldots, x_k, y \in \mathbb{S}_0$, $X \in \mathbb{S}_1$, $R \in \mathbb{S}_{k+1}$, and $\varphi, \varphi_1, \ldots, \varphi_k$ are formulas, for $k \geqslant 1$.		

Table 2.1. Syntax and semantics of the considered logics.

Class of formulas		*Generating grammar*
FOL	First-order	$\varphi ::= (x \doteq y) \mid X(x) \mid R(x_0, \ldots, x_k) \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \exists_x \varphi$
EMSOL	Existential MSOL	$\varphi ::= \psi \mid \exists_X \varphi$, where $\psi \in$ FOL. Equivalently, EMSOL $:= \Sigma_1^{\mathrm{MSO}}(\mathrm{FOL})$; see Section 6.1.
MSOL	Monadic second-order	$\varphi ::= (x \doteq y) \mid X(x) \mid R(x_0, \ldots, x_k) \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \exists_x \varphi \mid \exists_X \varphi$ Equivalently, MSOL $:=$ MSO(FOL).
$\overrightarrow{\mathrm{ML}}$	Modal	$\varphi ::= x \mid X \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \langle R \rangle(\varphi_1, \ldots, \varphi_k)$
$\overleftarrow{\mathrm{ML}}$	Backward modal	$\varphi ::= x \mid X \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \overleftarrow{\langle R \rangle}(\varphi_1, \ldots, \varphi_k)$
$\overleftrightarrow{\mathrm{ML}}$	Bidirectional modal	$\varphi ::= x \mid X \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \langle R \rangle(\varphi_1, \ldots, \varphi_k) \mid \overleftarrow{\langle R \rangle}(\varphi_1, \ldots, \varphi_k)$
$\overrightarrow{\mathrm{ML}}_g$	Modal with global modalities	$\varphi ::= x \mid X \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \langle R \rangle(\varphi_1, \ldots, \varphi_k) \mid \langle\cdot\rangle\varphi$
$\overleftarrow{\mathrm{ML}}_g$, $\overleftrightarrow{\mathrm{ML}}_g$		Analogous to the preceding grammars.
MSO(Φ)	Φ extended with set quantifiers	Same grammar as Φ with the additional choice "$\mid \exists_X \varphi$".
Here, $x, x_0, \ldots, x_k, y \in \mathbb{S}_0$, $X \in \mathbb{S}_1$, $R \in \mathbb{S}_{k+1}$, for $k \geqslant 1$, and $\Phi \in \{\overrightarrow{\mathrm{ML}}, \overleftarrow{\mathrm{ML}}, \ldots, \overleftrightarrow{\mathrm{ML}}_g, \mathrm{FOL}\}$.		

Table 2.2. The considered classes of formulas.

formula φ, i.e., outside the scope of a binding operator. If $\mathrm{free}(\varphi) \subseteq \sigma$, we say that φ is a *sentence* over σ. Sometimes, when the notions of “variable” and “constant” are clear from context, we also use the notation $\varphi(x_1, \dots, x_m, X_1, \dots, X_n)$ to indicate that at most the variables given in brackets occur freely in φ, i.e., that no other variables than $x_1, \dots, x_m, X_1, \dots, X_n$ lie in $\mathrm{free}(\varphi)$. The relation $\models$ defined in Table 2.1 specifies in which cases a structure G *satisfies* φ, written $G \models \varphi$, assuming that φ is a sentence over $\mathrm{sig}(G)$. Otherwise, we stipulate that $G \not\models \varphi$.

Of particular interest for this thesis are those formulas in which the node symbol @ is considered to be free, although it might not occur explicitly. They are evaluated on a pointed structure G from the perspective of the node $@^G$. Atomic formulas of the form x or X, with $x \in \mathbb{S}_0$ and $X \in \mathbb{S}_1$, are satisfied if $@^G$ is labeled by the corresponding symbol. Using the operator $\langle R \rangle$, which is called the R-*diamond*, we can remap the symbol @ through existential quantification over the nodes in G that are reachable from $@^G$ through the relation R^G. If we want to do the same with respect to the inverse relation of R^G, we can use the *backward* R-*diamond* $\overline{\langle R \rangle}$. In addition, there is also the *global diamond* $\langle \bullet \rangle$ (unfortunately often called “universal modality”), which ranges over all nodes of G. It can be considered as the diamond operator corresponding to the relation $V^G \times V^G$, i.e., the edge relation of the complete digraph over V^G. To facilitate certain descriptions, we shall sometimes treat $\overline{\langle R \rangle}$ and $\langle \bullet \rangle$ as special cases of $\langle R \rangle$, assuming that they are implicitly associated with the reserved relation symbols R^{-1} and $\bullet$, respectively. These symbols do not belong to $\mathbb{S}$, and therefore cannot be interpreted by any structure.

Allowing a bit of syntactic sugar, we will make liberal use of the remaining operators of predicate logic, i.e., $\wedge$, $\rightarrow$, $\leftrightarrow$, $\forall$, and we may leave out some parentheses, assuming that $\vee$ and $\wedge$ take precedence over $\rightarrow$ and $\leftrightarrow$. Furthermore, we define the abbreviations

$$\top \coloneqq @, \quad \bot \coloneqq \neg @, \quad \text{and}$$
$$[R](\varphi_1, \dots, \varphi_k) \coloneqq \neg \langle R \rangle(\neg\varphi_1, \dots, \neg\varphi_k).$$

Note that it makes sense to define $\top$ (“true”) as @, since by definition, the atomic formula @ is always satisfied at the point of evaluation. Also, the second line remains applicable if one substitutes R^{-1} or $\bullet$ for R. The resulting operators $[R]$, $\overline{[R]}$ and $[\bullet]$ provide universal quantification and are called *boxes* (using the same attributes as for diamonds). Diamonds and boxes are collectively referred to as *modalities* or *modal operators*. In case we restrict ourselves to structures that only have a single relation, we may omit the relation symbol R, and just use empty modalities such as $\Diamond$. Similarly, if the relation symbols involved are indexed, like $R_1, \dots, R_r$, we associate them with modalities of the form $\langle i \rangle$, for $1 \leqslant i \leqslant r$.

Let us now turn to the particular classes of formulas considered in this thesis, which are specified in Table 2.2. The languages of *first-order logic* (FOL), *existential monadic second-order logic* (EMSOL), and *monadic second-order logic* (MSOL) are defined in the usual way. When evaluated on some structure G, their atomic formulas allow to compare nodes assigned to node symbols in $\mathrm{sig}(G)$ with respect to the equality relation and any other relation assigned to a relation symbol in $\mathrm{sig}(G)$. In FOL, we can assign new interpretations to node symbols by means of existential and universal *quantification over nodes*. In EMSOL, we may additionally reinterpret set symbols using existential *quantifiers over sets* of nodes, and in MSOL, we can also use the corresponding universal quantifiers.

The remaining classes of formulas can all be qualified as modal languages, insofar as they include modal operators instead of the classical first-order quantifiers. By performing this change of paradigm, we lose our “bird’s-eye view” of the structure G, and now see it from the local point of view of the node $@^G$. (For this, G obviously has to be pointed.) In basic *modal logic* ($\overrightarrow{\text{ML}}$), a node “sees” only its outgoing neighbors, and thus our domain of quantification is restricted to those neighbors. Furthermore, the position symbol $@$ is the only node symbol whose interpretation can be changed by a modal operator. *Backward modal logic* ($\overleftarrow{\text{ML}}$) is the variant of $\overrightarrow{\text{ML}}$ where a node “sees” its incoming neighbors instead of its outgoing neighbors, whereas *bidirectional modal logic* ($\overleftrightarrow{\text{ML}}$) is the combination where a node “sees” both incoming and outgoing neighbors. We will also look at *modal logic with global modalities* ($\overrightarrow{\text{ML}}_{\text{g}}$), where we regain the possibility to quantify over the entire domain of the structure, but are still confined to remapping only the position symbol $@$. The backward and bidirectional variants $\overleftarrow{\text{ML}}_{\text{g}}$ and $\overleftrightarrow{\text{ML}}_{\text{g}}$ are defined analogously. Finally, we also consider crossover versions of modal logic that are enriched with the set quantifiers of MSOL. Given a class of formulas Φ, we denote by $\text{MSO}(\Phi)$ the corresponding enriched class. For instance, the formulas of $\text{MSO}(\overrightarrow{\text{ML}})$ are generated by the grammar

$$\varphi ::= x \mid X \mid \neg\varphi \mid (\varphi_1 \vee \varphi_2) \mid \Diamond_{R}(\varphi_1, \dots, \varphi_k) \mid \exists X \varphi,$$

where $x \in \mathbb{S}_0$, $X \in \mathbb{S}_1$, and $R \in \mathbb{S}_{k+1}$. Note that by this notation, $\text{MSOL} = \text{MSO}(\text{FOL})$.

For any class of formulas Φ, we shall refer to its members as *Φ-formulas*. Given a Φ-formula φ, a class of structures $\mathcal{C}$ (e.g., DG), and a structure G, we use the semantic bracket notations $[\![\varphi]\!]_{\mathcal{C}}$ and $[\![\varphi]\!]_G$ to denote the structure language *defined* by φ over $\mathcal{C}$, and the set of nodes of G at which φ holds. More formally,

$$[\![\varphi]\!]_{\mathcal{C}} := \{G \in \mathcal{C} \mid G \vDash \varphi\}, \quad \text{and}$$
$$[\![\varphi]\!]_G := \{v \in V^G \mid G[@ \mapsto v] \vDash \varphi\}.$$

Furthermore, $[\![\Phi]\!]_{\mathcal{C}}$ denotes the family of structure languages that are *definable* in Φ (or *Φ-definable*) over $\mathcal{C}$, i.e.,

$$[\![\Phi]\!]_{\mathcal{C}} := \{[\![\varphi]\!]_{\mathcal{C}} \mid \varphi \in \Phi\}.$$

If $\mathcal{C}$ is equal to the set of all structures, then we do not have to specify it explicitly as a subscript; that is, we may simply write $[\![\varphi]\!]$ and $[\![\Phi]\!]$ instead of $[\![\varphi]\!]_{\mathcal{C}}$ and $[\![\Phi]\!]_{\mathcal{C}}$. Similarly, we use

$$[\varphi]_{\mathcal{C}} := \{\psi \mid [\![\psi]\!]_{\mathcal{C}} = [\![\varphi]\!]_{\mathcal{C}}\}$$

for the equivalence class of φ over $\mathcal{C}$, and

$$[\Phi]_{\mathcal{C}} := \bigcup_{\varphi \in \Phi} [\varphi]_{\mathcal{C}}$$

for the set of all formulas that are equivalent over $\mathcal{C}$ to some formula in Φ. Again, we may drop the subscript if we do not want to restrict to a particular class of structures.

2.6 Example formulas

In order to illustrate the syntax introduced in the previous section, we now look at two simple examples.

The first is a great classic that is often used to show how a widely known graph property can be expressed in MSOL without too much effort.

Example 2.1 (3-Colorability).
▶ The following EMSOL-formula defines the language of 3-colorable digraphs over DG.

$$
\begin{aligned}
\varphi_3^{\text{color}} := \exists_{X_1,X_2,X_3}\Big(\ \forall_x\Big(\ &\big(X_1(x) \vee X_2(x) \vee X_3(x)\big) \wedge \\
&\neg\big(X_1(x) \wedge X_2(x)\big) \wedge \\
&\neg\big(X_1(x) \wedge X_3(x)\big) \wedge \\
&\neg\big(X_2(x) \wedge X_3(x)\big) \qquad \Big) \wedge \\
\forall_{x,y}\Big(R(x,y) \ \to\ &\neg\big(X_1(x) \wedge X_1(y)\big) \wedge \\
&\neg\big(X_2(x) \wedge X_2(y)\big) \wedge \\
&\neg\big(X_3(x) \wedge X_3(y)\big) \qquad \Big)\Big)
\end{aligned}
$$

The existentially quantified set variables $X_1, X_2, X_3 \in \mathbb{S}_1$ represent the three possible colors. In the first four lines, we specify that the sets assigned to these variables form a partition of the set of nodes (possibly with empty components). The remaining three lines constitute the actual definition of a valid coloring: no two adjacent nodes share the same color, which means that adjacent nodes are in different sets. ◀

Our second example is equally simple, but less glamorous because it illustrates a technical issue that will concern us in Chapter 6, where we shall work with $\text{MSO}(\overrightarrow{\text{ML}}_g)$ and some variants thereof. As we do not allow first-order quantification in modal logic with set quantifiers, some properties that seem very natural in FOL (and thus MSOL) become rather cumbersome to express. Nevertheless, translation from FOL to $\text{MSO}(\overrightarrow{\text{ML}}_g)$ is always possible because we can simulate first-order quantifiers by set quantifiers relativized to singletons, which, by extension, also entails the equivalence of MSOL and $\text{MSO}(\overrightarrow{\text{ML}}_g)$. Example 2.2 presents the basic construction that allows us to do this. We will refer to it several times in Chapter 6.

Example 2.2 (Uniqueness).
▶ Consider the following formula schema, where $X \in \mathbb{S}_1$, $R \in \mathbb{S}_2$, and φ can be any $\overleftrightarrow{\text{ML}}_g$-formula:

$$\text{see1}_R(\varphi) := \langle R\rangle\varphi \ \wedge\ \forall_X\big(\langle R\rangle(\varphi \wedge X) \to [R](\varphi \to X)\big).$$

When evaluated on a pointed structure G whose signature includes $\{@, R\} \cup \text{free}(\varphi)$, the formula $\text{see1}_R(\varphi)$ states that there is exactly one node $v \in V^G$ reachable from $@^G$ through an R^G-edge, such that φ is satisfied at v (i.e., by the structure $G[@ \mapsto v]$). In the context of 1-relational digraphs, we may use the shorthand $\text{see1}(\varphi)$ to invoke this schema. Using the same construction with global modalities, we also define

$$\text{tot1}(\varphi) := \text{see1}_\bullet(\varphi),$$

which states that there is precisely one node in the entire structure G at which φ is satisfied. Here, G does not necessarily have to be pointed, and, of course, $\text{sig}(G)$ does not contain $\bullet$ (since it is the symbol reserved for the total symmetric relation). ◀

Anticipating the notation of Section 6.1, the formulas obtained by the construction in Example 2.2 can be classified as $[\Pi_1^{\text{MSO}}(\Phi)]$-formulas, where $\Phi \in \{\overrightarrow{\text{ML}}, \overleftrightarrow{\text{ML}}, \overrightarrow{\text{ML}}_g, \overleftrightarrow{\text{ML}}_g\}$ depends on the specific modalities that occur in φ.

2.7 Distributed automata

We conclude this preliminary chapter by introducing our primary objects of interest. Simply put, a distributed automaton is a deterministic finite-state machine $\mathcal{A}$ that reads sets of states instead of the usual alphabetic symbols. To run $\mathcal{A}$ on a 1-relational digraph G, we place a separate copy of the machine on every node v of G, initialize it to a state that may depend on v's label $\lambda^G(v)$, and then let all the nodes communicate in an infinite sequence of synchronous rounds. In every round, each node computes its next state as a function of its own current state and the set of states of its incoming neighbors. Intuitively, node v broadcasts its current state q to every outgoing neighbor, while at the same time collecting the states received from its incoming neighbors into a set S; the successor state of q is then computed as a function of q and S. Since S is a set (as opposed to a multiset or a vector), v cannot distinguish between two incoming neighbors that share the same state. Now, acting as a semi-decider, the machine at node v accepts precisely if it visits an accepting state at some point in time. Either way, all machines of the network keep running and communicating forever. This is because even if a node has already accepted, it may still obtain new information that affects the acceptance behavior of its outgoing neighbors.

Let us now define the notion of distributed automaton more formally, and generalize it to digraphs with an arbitrary number of edge relations.

Definition 2.3 (Distributed automaton).
▸ A (deterministic, *nonlocal*) *distributed automaton* (DA) over Σ-labeled, r-relational digraphs is a tuple $\mathcal{A} = (Q, \delta_0, \delta, F)$, where Q is a finite nonempty set of *states*, $\delta_0 \colon \Sigma \to Q$ is an *initialization function*, $\delta \colon Q \times (2^Q)^r \to Q$ is a *transition function*, and $F \subseteq Q$ is a set of *accepting states*. ◂

Let G be a Σ-labeled, r-relational digraph. The *run* of $\mathcal{A}$ on G is an infinite sequence $\rho = (\rho_0, \rho_1, \rho_2, \dots)$ of maps $\rho_t \colon V^G \to Q$, called *configurations*, which are defined inductively as follows, for $t \in \mathbb{N}$ and $v \in V^G$:

$$\rho_0(v) = \delta_0(\lambda^G(v)) \qquad \text{and} \qquad \rho_{t+1}(v) = \delta\Big(\rho_t(v), \big(\{\rho_t(u) \mid uv \in R_i^G\}\big)_{1 \leqslant i \leqslant r}\Big).$$

For $v \in V^G$, the automaton $\mathcal{A}$ *accepts* the pointed digraph $G[v]$ if v visits an accepting state at some point in the run ρ of $\mathcal{A}$ on G, i.e., if there exists $t \in \mathbb{N}$ such that $\rho_t(v) \in F$. The pointed-digraph language of $\mathcal{A}$, or pointed-digraph language *recognized* by $\mathcal{A}$, is the set of all pointed digraphs that are accepted by $\mathcal{A}$. We denote this language by $[\![\mathcal{A}]\!]_{@\text{DG}_\Sigma^r}$, in analogy to our notation for logical formulas. Similarly, given a class of automata $\mathcal{A}$, we write $[\![\mathcal{A}]\!]_{@\text{DG}_\Sigma^r}$ for the class of pointed-digraph languages over $@\text{DG}_\Sigma^r$ that are recognized by some member of $\mathcal{A}$; we call them *$\mathcal{A}$-recognizable*.

As usual, two devices (i.e., automata or formulas) are *equivalent* if they specify (i.e., recognize or define) the same language.

In distributed computing, one often considers algorithms that run in a constant number of synchronous rounds. They are known as *local algorithms* (see, e.g., [Suo13]). Here, we use the same terminology for distributed automata and give a syntactic definition of locality in terms of state diagrams. Basically, a distributed automaton is local if its state diagram does not contain any directed cycles, except for self-loops on sink states. This is equivalent to requiring that all nodes stop changing their state after a constant number of rounds.

Definition 2.4 (Local distributed automaton).
▶ A *local distributed automaton* (LDA) over r-relational digraphs is a distributed automaton $A = (Q, \delta_0, \delta, F)$ whose state diagram satisfies the following two conditions:

a. The only directed cycles are self-loops. That is, for every sequence $q_1, q_2, \dots, q_n$ of states in Q such that $q_1 = q_n$ and $\delta(q_i, \vec{S}_i) = q_{i+1}$ for some $\vec{S}_i \in (2^Q)^r$, it must be that all states of the sequence are the same.

b. Self-loops occur only on sink states. That is, for every state $q \in Q$, if $\delta(q, \vec{S}) = q$ for some $\vec{S} \in (2^Q)^r$, then the same must hold for all $\vec{S} \in (2^Q)^r$. ◀

Deviating only in nonessential details from the original presentation given by Hella et al. in [HJK⁺12, HJK⁺15], we can now restate their logical characterization of the class SB(1) using the terminology introduced above.

Theorem 2.5 ($[\![\text{LDA}]\!]_{@\text{DG}^r_\Sigma} = [\![\overleftarrow{\text{ML}}]\!]_{@\text{DG}^r_\Sigma}$; [HJK⁺12, HJK⁺15]).
▶ A pointed-digraph language is recognizable by a local distributed automaton if and only if it is definable by a formula of backward modal logic. There are effective translations in both directions. ◀

The notion of locality plays a major role in Chapter 3, where we extend LDA's with the capacity of alternation and a global acceptance condition. Our extension leaves the realm of basic DA's, since we show that it is equivalent to MSOL, which by [Kuu13a, Prp. 6 & 8] is incomparable with DA's.

On the other hand, in Chapters 4 and 5, we consider a simpler extension of LDA's, which can be seen as a natural intermediate stage between LDA's and DA's. Given the above definition of local automata, a rather obvious generalization is to allow self-loops on all states, even if they are not sink states; we call this property quasi-acyclic. More formally, a *quasi-acyclic distributed automaton* (QDA) is a DA that satisfies Item a of Definition 2.4, but not necessarily Item b. An example of such an automaton will be provided in Section 4.1 (Figure 4.1 on page 40).

Chapter based on the conference paper [Rei15].

3
Alternating Local Automata

In this chapter, we transfer the well-established notion of alternating automaton to the setting of local distributed automata and combine it with a global acceptance condition. This gives rise to a new class of graph automata that recognize precisely the languages of finite digraphs definable in MSOL. By restricting transitions to be nondeterministic or deterministic, we also obtain two strictly weaker variants for which the emptiness problem is decidable.

3.1 Informal description

We start with an informal description of the adjustments that we make to the basic model of local automata (see Section 2.7). Formal definitions will follow in Section 3.2.

The term "*local distributed automaton with global acceptance condition*" (LDA_g) will be used to refer collectively to the deterministic, nondeterministic and alternating versions of our model. Let us first mention the properties they have in common.

Levels of states. As for basic local automata, the number of communication rounds is limited by a constant. To make this explicit and to simplify the subsequent definition of alternation, we associate a number, called level, with every state. In most cases, this number indicates the round in which the state may occur. We require that potentially initial states are at level 0, and outgoing transitions from states at level i go to states at level $i + 1$. There is an exception, however: the states at the highest level, called the permanent states, can also be initial states and can have incoming transitions from any level. Moreover, all their outgoing transitions are self-loops. The idea is that, once a node has reached a permanent state, it terminates its local computation, and waits for the other nodes in the digraph to terminate too.

Global acceptance. Unlike for basic local automata, the considered input is a digraph, not a pointed digraph, and consequently the language recognized by an LDA_g is a digraph language. For this reason, once all the nodes have reached a permanent state, the LDA_g ceases to operate as a distributed algorithm, and collects all the reached permanent states into a set F. This set is the sole acceptance criterion: if F

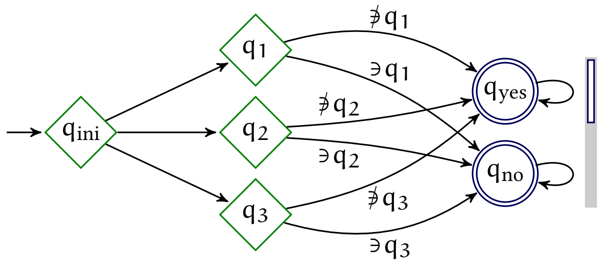

Figure 3.1. A_3^{color}, a nondeterministic LDA$_{\mathrm{g}}$ over unlabeled, 1-relational digraphs whose digraph language consists of the 3-colorable digraphs.

is part of the LDA$_{\mathrm{g}}$'s accepting sets, then the input digraph is accepted, otherwise it is rejected. In particular, the automaton cannot detect whether several nodes have reached the same permanent state. This limitation is motivated by the desire to have a simple finite representation of LDA$_{\mathrm{g}}$'s; in other words, the same reason why we do not allow nodes to distinguish between several neighbors that are in the same state.

As an introductory example, we translate the MSOL-formula $\varphi_3^{\mathrm{color}}$ from Example 2.1 in Section 2.6 to the setting of LDA$_{\mathrm{g}}$'s.

Example 3.1 (3-colorability)**.**
▶ Figure 3.1 shows the state diagram of a simple nondeterministic LDA$_{\mathrm{g}}$ A_3^{color}. The states are arranged in columns corresponding to their levels, ascending from left to right. A_3^{color} expects an unlabeled digraph as input, and accepts it if and only if it is 3-colorable. The automaton proceeds as follows: All nodes of the input digraph are initialized to the state q_{ini}. In the first round, each node nondeterministically chooses to go to one of the states q_1, q_2 and q_3, which represent the three possible colors. Then, in the second round, the nodes verify locally that the chosen coloring is valid. If the set received from their incoming neighborhood (only one, since there is only a single edge relation) contains their own state, they go to q_{no}, otherwise to q_{yes}. The automaton then accepts the input digraph if and only if all the nodes are in q_{yes}, i.e., $\{q_{\mathrm{yes}}\}$ is its only accepting set. This is indicated by the bar to the right of the state diagram. We shall refer to such a representation of sets using bars as *barcode*. ◀

The last property, which applies only to our most powerful version of LDA$_{\mathrm{g}}$'s, is *alternation*, a generalization of nondeterminism introduced by Chandra, Kozen and Stockmeyer in [CKS81] (there, for Turing machines and other types of word automata).

Alternation. In addition to being able to nondeterministically choose between different transitions, nodes can also explore several choices in parallel. To this end, the nonpermanent states of an alternating LDA$_{\mathrm{g}}$ (ALDA$_{\mathrm{g}}$) are partitioned into two types, existential and universal, such that states on the same level are of the same type. If, in a given round, the nodes are in existential states, then they nondeterministically choose a single state to go to in the next round, as described above. In contrast, if they are in universal states, then the run of the ALDA$_{\mathrm{g}}$ is split into several parallel branches, called universal branches, one for each possible combination of choices of the nodes. This procedure of splitting is repeated recursively for each round in which the nodes are in universal states. The ALDA$_{\mathrm{g}}$ then accepts the input digraph if and only if its acceptance condition is satisfied in every universal branch of the run.

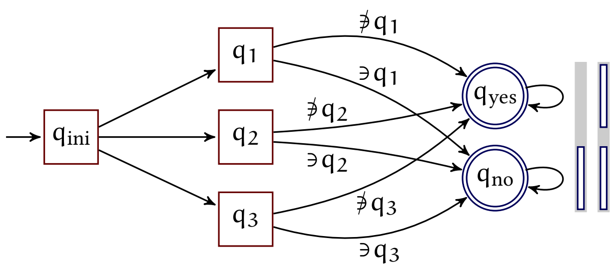

Figure 3.2. $\overline{A}_3^{color}$, an alternating LDA$_g$ over unlabeled, 1-relational digraphs whose digraph language consists of the digraphs that are *not* 3-colorable.

Example 3.2 (Non-3-colorability).
▸ To illustrate the notion of universal branching, consider the ALDA$_g$ $\overline{A}_3^{color}$ shown in Figure 3.2. It is a complement automaton of A_3^{color} from Example 3.1, i.e., it accepts precisely those (unlabeled) digraphs that are *not* 3-colorable. States represented as boxes are universal (whereas the diamonds in Figure 3.1 stand for existential states). Given an input digraph with n nodes, $\overline{A}_3^{color}$ proceeds as follows: All nodes are initialized to q_{ini}. In the first round, the run is split into 3^n universal branches, each of which corresponds to one possible outcome of the first round of A_3^{color} running on the same input digraph. Then, in the second round, in each of the 3^n universal branches, the nodes check whether the coloring chosen in that branch is valid. As indicated by the barcode, the acceptance condition of $\overline{A}_3^{color}$ is satisfied if and only if at least one node is in state q_{no}, i.e., the accepting sets are $\{q_{no}\}$ and $\{q_{yes}, q_{no}\}$. Hence, the automaton accepts the input digraph if and only if no valid coloring was found in any universal branch. Note that we could also have chosen to make the states q_1, q_2 and q_3 existential, since their outgoing transitions are deterministic. Regardless of their type, there is no branching in the second round. ◂

3.2 Formal definitions

We now repeat and clarify the notions from Section 3.1 in a more formal setting, beginning with our most general definition of LDA$_g$'s.

Definition 3.3 (Alternating local distributed automaton).
▸ An *alternating local distributed automaton with global acceptance condition* (ALDA$_g$) over Σ-labeled, r-relational digraphs is a tuple $A = (\vec{Q}, \delta_0, \delta, \mathcal{F})$, where

- $\vec{Q} = (Q_\exists, Q_\forall, Q_P)$ is a collection of states, with $Q_\exists$, $Q_\forall$ and $Q_P \neq \varnothing$ being pairwise disjoint finite sets of *existential*, *universal* and *permanent states*, respectively, also referred to by the notational shorthands
  - $Q \coloneqq Q_\exists \cup Q_\forall \cup Q_P$, for the entire set of *states*,
  - $Q_N \coloneqq Q_\exists \cup Q_\forall$, for the set of *nonpermanent states*,
- $\delta_0 \colon \Sigma \to Q$ is an *initialization function*,
- $\delta \colon Q \times (2^Q)^r \to 2^Q$ is a (local) *transition function*, and
- $\mathcal{F} \subseteq 2^{Q_P}$ is a set of *accepting sets* of permanent states.

The functions δ_0 and δ must be such that one can unambiguously associate with every state $q \in Q$ a *level* $l_A(q) \in \mathbb{N}$ satisfying the following conditions:

- States on the same level are of the same type, i.e., for every $i \in \mathbb{N}$,
$$\{q \in Q \mid l_A(q) = i\} \in (2^{Q_\exists} \cup 2^{Q_\forall} \cup 2^{Q_P}).$$
- Initial states are either on the lowest level or permanent, i.e., for every $q \in Q$,
$$\exists a \in \Sigma \colon \delta_0(a) = q \quad \text{implies} \quad l_A(q) = 0 \lor q \in Q_P.$$
- Nonpermanent states without incoming transitions are on the lowest level, and transitions between nonpermanent states go only from one level to the next, i.e., for every $q \in Q_N$,
$$l_A(q) = \begin{cases} 0 & \text{if for all } p \in Q \text{ and } \vec{S} \in (2^Q)^r, \\ & \text{it holds that } q \notin \delta(p, \vec{S}), \\ i+1 & \text{if there are } p \in Q_N \text{ and } \vec{S} \in (2^Q)^r \\ & \text{such that } l_A(p) = i \text{ and } q \in \delta(p, \vec{S}). \end{cases}$$
- The permanent states are one level higher than the highest nonpermanent ones, and have only self-loops as outgoing transitions, i.e., for every $q \in Q_P$,
$$l_A(q) = \begin{cases} 0 & \text{if } Q_N = \varnothing, \\ \max\{l_A(q) \mid q \in Q_N\} + 1 & \text{otherwise,} \end{cases}$$
$$\delta(q, \vec{S}) = \{q\} \quad \text{for every } \vec{S} \in (2^Q)^r.$$ ◀

For any ALDA_g $A = (\vec{Q}, \delta_0, \delta, \mathcal{F})$, we define its *length* $\text{len}(A)$ to be its highest level, that is, $\text{len}(A) := \max\{l_A(q) \mid q \in Q\}$.

Next, we want to give a formal definition of a run. For this, we need the notion of a configuration, which can be seen as the global state of an ALDA_g.

Definition 3.4 (Configuration).
▶ Consider a digraph G and an ALDA_g $A = (\vec{Q}, \delta_0, \delta, \mathcal{F})$. For any map $\zeta \colon V^G \to Q$, we call the Q-labeled variant $G[\zeta]$ of G a *configuration* of A on G. If every node in $G[\zeta]$ is labeled by a permanent state, we refer to that configuration as a *permanent configuration*. Otherwise, if $G[\zeta]$ is a nonpermanent configuration whose nodes are labeled exclusively by existential and (possibly) permanent states, we say that $G[\zeta]$ is an *existential configuration*. Analogously, the configuration is *universal* if it is nonpermanent and only labeled by universal and (possibly) permanent states.

Additionally, we say that a permanent configuration $G[\zeta]$ is *accepting* if the set of states occurring in it is accepting, i.e., if $\{\zeta(v) \mid v \in V^G\} \in \mathcal{F}$. Any other permanent configuration is called *rejecting*. Nonpermanent configurations are neither accepting nor rejecting. ◀

The (local) transition function of an ALDA_g specifies for each state a set of potential successors, for a given family of sets of states. This can be naturally extended to configurations, which leads us to the definition of a global transition function.

Definition 3.5 (Global transition function).
▸ The *global transition function* δ_g of an ALDA$_g$ $\mathcal{A} = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over Σ-labeled, r-relational digraphs assigns to each configuration $G[\zeta]$ of $\mathcal{A}$ the set of all of its *successor configurations* $G[\eta]$, by combining all possible outcomes of local transitions on $G[\zeta]$, i.e.,

$$\delta_g \colon \mathrm{DG}^r_Q \to 2^{(\mathrm{DG}^r_Q)}$$
$$G[\zeta] \mapsto \Big\{ G[\eta] \;\Big|\; \bigwedge_{v \in V^G} \eta(v) \in \delta\Big(\zeta(v), \big(\{\zeta(u) \mid uv \in R^G_i\}\big)_{i \in [r]}\Big)\Big\}. \qquad ◂$$

We now have everything at hand to formalize the notion of a run.

Definition 3.6 (Run).
▸ A *run* of an ALDA$_g$ $\mathcal{A} = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over Σ-labeled, r-relational digraphs on a given digraph $G \in \mathrm{DG}^r_\Sigma$ is an acyclic digraph ρ whose nodes are configurations of $\mathcal{A}$ on G, such that

- the *initial configuration* $G[\delta_0 \circ \lambda^G] \in V^\rho$ is the only source,
- every nonpermanent configuration $G[\zeta] \in V^\rho$ with set of successor configurations $\delta_g(G[\zeta]) = \{G[\eta_1], \dots, G[\eta_m]\}$ has
  - exactly one outgoing neighbor $G[\eta_i] \in \delta_g(G[\zeta])$ if $G[\zeta]$ is existential,
  - exactly m outgoing neighbors $G[\eta_1], \dots, G[\eta_m]$ if $G[\zeta]$ is universal, and
- every permanent configuration $G[\zeta] \in V^\rho$ is a sink.

Here, the operator $\circ$ denotes standard function composition, such that $(\delta_0 \circ \lambda^G)(v) = \delta_0(\lambda^G(v))$.

The run ρ is *accepting* if every permanent configuration $G[\zeta] \in V^\rho$ is accepting. ◂

An ALDA$_g$ $\mathcal{A} = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over Σ-labeled, r-relational digraphs *accepts* a given digraph $G \in \mathrm{DG}^r_\Sigma$ if and only if there exists an accepting run ρ of $\mathcal{A}$ on G. The digraph language *recognized* by $\mathcal{A}$ is the set

$$[\![\mathcal{A}]\!]_{\mathrm{DG}^r_\Sigma} \coloneqq \big\{ G \in \mathrm{DG}^r_\Sigma \;\big|\; \mathcal{A} \text{ accepts } G \big\}.$$

A digraph language that is recognized by some ALDA$_g$ is called *ALDA$_g$-recognizable*. We denote by $[\![\mathrm{ALDA_g}]\!]_{\mathrm{DG}^r_\Sigma}$ the class of all such digraph languages.

The ALDA$_g$ $\mathcal{A}$ is equivalent to some MSOL-formula φ if it recognizes precisely the digraph language defined by φ over DG^r_Σ, i.e., if $[\![\mathcal{A}]\!]_{\mathrm{DG}^r_\Sigma} = [\![\varphi]\!]_{\mathrm{DG}^r_\Sigma}$.

We inductively define that a configuration $G[\zeta] \in \mathrm{DG}^r_Q$ is *reachable* by $\mathcal{A}$ on G if either $G[\zeta] = G[\delta_0 \circ \lambda^G]$, or $G[\zeta] \in \delta_g(G[\eta])$ for some configuration $G[\eta] \in \mathrm{DG}^r_Q$ reachable by $\mathcal{A}$ on G. In case G is irrelevant, we simply say that $G[\zeta]$ is reachable by $\mathcal{A}$.

The automaton $\mathcal{A}$ is called a *nondeterministic* LDA$_g$ (NLDA$_g$) if it has no universal states, i.e., if $Q_\forall = \varnothing$. If additionally every configuration $G[\zeta] \in \mathrm{DG}^r_Q$ that is reachable by $\mathcal{A}$ has precisely one successor configuration, i.e., $\big|\delta_g(G[\zeta])\big| = 1$, then we refer to $\mathcal{A}$ as a *deterministic* LDA$_g$ (DLDA$_g$). We denote the classes of NLDA$_g$- and DLDA$_g$-recognizable digraph languages by $[\![\mathrm{NLDA_g}]\!]_{\mathrm{DG}^r_\Sigma}$ and $[\![\mathrm{DLDA_g}]\!]_{\mathrm{DG}^r_\Sigma}$.

Let us now illustrate the notion of ALDA$_g$ using a slightly more involved example.

Example 3.7 (Concentric circles).
▶ Consider the ALDA$_g$ $\mathcal{A}_{\text{centric}} = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over $\{a, b, c\}$-labeled digraphs represented by the state diagram in Figure 3.3. Again, existential states are represented by diamonds, universal states by boxes, and permanent states by double circles. The short arrows mapping node labels to states indicate the initialization function δ_0. For instance, $\delta_0(a) = q_a$. The other arrows specify the transition function δ. A label on such a transition arrow indicates a requirement on the set of states that a node receives from its incoming neighborhood (only one set, since there is only a single edge relation). For instance, $\delta\big(q_b, (\{q_a, q_c\})\big) = \{q_{b:1}, q_{b:2}\}$. If there is no label, any set is permitted. Finally, as indicated by the barcode on the far right, the set of accepting sets is $\mathcal{F} = \big\{\{q_{a:3}, q_{yes}\}, \{q_{a:4}, q_{yes}\}\big\}$.

Intuitively, $\mathcal{A}_{\text{centric}}$ proceeds as follows: In the first round, the a-labeled nodes do nothing but update their state, while the b- and c-labeled nodes verify that the labels in their incoming neighborhood satisfy the condition of a valid graph coloring. The c-labeled nodes additionally check that they do not see any a's, and then directly terminate. Meanwhile, the b-labeled nodes nondeterministically choose one of the markers 1 and 2. In the second round, only the a-labeled nodes are busy. They verify that their incoming neighborhood consists exclusively of b-labeled nodes, and that both of the markers 1 and 2 are present, thus ensuring that they have at least two incoming neighbors. Then, they simultaneously pick the markers 3 and 4, thereby creating different universal branches, and the run of the automaton terminates. Finally, the ALDA$_g$ checks that all the nodes approve of the digraph (meaning that none of them has reached the state q_{no}), and that in each universal branch, precisely one of the markers 3 and 4 occurs, which implies that there is a unique a-labeled node.

To sum up, the digraph language $[\![\mathcal{A}_{\text{centric}}]\!]_{\mathrm{DG}^r_\Sigma}$ consists of all the $\{a, b, c\}$-labeled, digraphs such that

- the labeling constitutes a valid 3-coloring,
- there is precisely one a-labeled node v_a, and
- v_a has only b-labeled nodes in its undirected neighborhood, and at least two incoming neighbors.

The name "$\mathcal{A}_{\text{centric}}$" refers to the fact that, in the (weakly) connected component of v_a, the b- and c-labeled nodes form con*centric* circles around v_a, i.e., nodes at distance 1 of v_a are labeled with b, nodes at distance 2 (if existent) with c, nodes at distance 3 (if existent) with b, and so forth.

Figure 3.4 shows an example of a labeled digraph that lies in $[\![\mathcal{A}_{\text{centric}}]\!]_{\mathrm{DG}^r_\Sigma}$. A corresponding accepting run can be seen in Figure 3.5. The leftmost configuration is existential, the next one is universal, and the two double-circled ones are permanent. In the first round, the three nodes that are in state q_b have a nondeterministic choice between $q_{b:1}$ and $q_{b:2}$. Hence, the second configuration is one of eight possible choices. The branching in the second round is due to the node in state q'_a which goes simultaneously to $q_{a:3}$ and $q_{a:4}$. In both branches, an accepting configuration is reached, since $\{q_{a:3}, q_{yes}\}$ and $\{q_{a:4}, q_{yes}\}$ are both accepting sets. Therefore, the entire run is accepting. ◀

In the following subsections (3.3, 3.4 and 3.5), we derive our results on several properties of LDA$_g$'s.

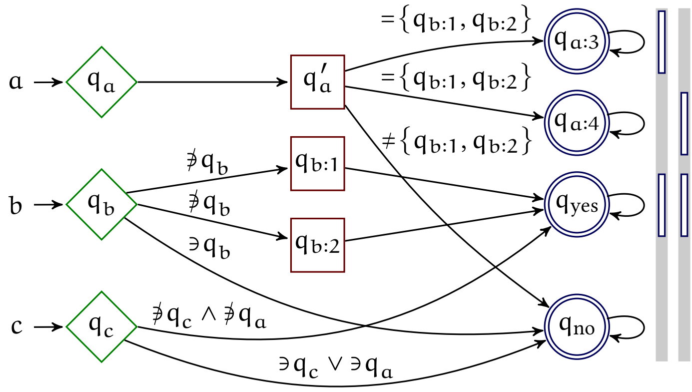

Figure 3.3. A_{centric}, an ALDA$_g$ over $\{a, b, c\}$-labeled digraphs whose digraph language consists of the labeled digraphs that satisfy the following conditions: the labeling constitutes a valid 3-coloring, there is precisely one a-labeled node v_a, the undirected neighborhood of v_a contains only b-labeled nodes, and v_a has at least two incoming neighbors.

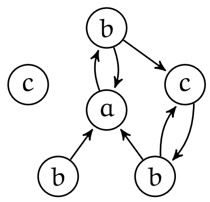

Figure 3.4. An $\{a, b, c\}$-labeled, digraph.

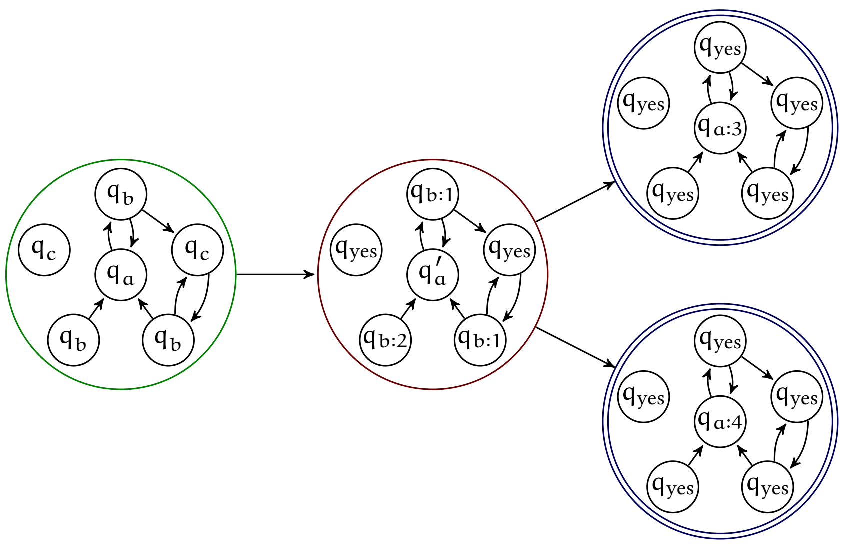

Figure 3.5. An accepting run of the ALDA$_g$ of Figure 3.3 on the labeled digraph shown in Figure 3.4.

3.3 Hierarchy and closure properties

By a (node) *projection* we mean a mapping $h\colon \Sigma \to \Sigma'$ between two alphabets Σ and Σ'. With slight abuse of notation, such a mapping is extended to labeled digraphs by applying it to each node label, and to digraph languages by applying it to each labeled digraph. That is, for every $G \in \mathrm{DG}_\Sigma^r$ and $L \subseteq \mathrm{DG}_\Sigma^r$,

$$h(G) := G[h \circ \lambda^G], \quad \text{and} \quad h(L) := \{h(G) \mid G \in L\},$$

where the operator $\circ$ denotes function composition, such that $(h \circ \lambda^G)(v) = h(\lambda^G(v))$.

Proposition 3.8 (Closure properties of $[\![\mathrm{ALDA_g}]\!]_{\mathrm{DG}_\Sigma^r}$).
▶ The class $[\![\mathrm{ALDA_g}]\!]_{\mathrm{DG}_\Sigma^r}$ of $\mathrm{ALDA_g}$-recognizable digraph languages is effectively closed under Boolean set operations and under projection. ◀

Proof sketch. As usual for alternating automata, complementation can be achieved by simply swapping the existential and universal states, and complementing the acceptance condition. That is, for an $\mathrm{ALDA_g}$ $A = \big((Q_\exists, Q_\forall, Q_P), \delta_0, \delta, \mathcal{F}\big)$ over Σ-labeled, r-relational digraphs, a complement automaton is $\overline{A} = \big((Q_\forall, Q_\exists, Q_P), \delta_0, \delta, 2^{Q_P} \setminus \mathcal{F}\big)$. This can be easily seen by associating a two-player game with A and any Σ-labeled, r-relational digraph G. One player tries to come up with an accepting run of A on G, whereas the other player seeks to find a (path to a) rejecting configuration in any run proposed by the adversary. The first player has a winning strategy if and only if A accepts G. (This game-theoretic characterization will be used and explained more extensively in the proof of Theorem 3.13.) From this perspective, the construction of $\overline{A}$ corresponds to interchanging the roles and winning conditions of the two players.

For two $\mathrm{ALDA_g}$'s A_1 and A_2, we can effectively construct an $\mathrm{ALDA_g}$ $A_\cup$ that recognizes $[\![A_1]\!]_{\mathrm{DG}_\Sigma^r} \cup [\![A_2]\!]_{\mathrm{DG}_\Sigma^r}$ by taking advantage of nondeterminism. The approach is, in principle, very similar to the corresponding construction for nondeterministic finite automata on words. In the first round of $A_\cup$, each node in the input digraph nondeterministically and independently decides whether to behave like in A_1 or in A_2. If there is a consensus, then the run continues as it would in the unanimously chosen automaton A_j, and it is accepting if and only if it corresponds to an accepting run of A_j. Otherwise, a conflict is detected, either locally by adjacent nodes that have chosen different automata, or at the latest, when acceptance is checked globally (important for disconnected digraphs), and in either case the run is rejecting. (Note that we have omitted some technicalities that ensure that the construction outlined above satisfies all the properties of an $\mathrm{ALDA_g}$.)

Closure under node projection is straightforward, again by exploiting nondeterminism. Given an $\mathrm{ALDA_g}$ A with node alphabet Σ and a projection $h\colon \Sigma \to \Sigma'$, we can effectively construct an $\mathrm{ALDA_g}$ A' that recognizes $h([\![A]\!]_{\mathrm{DG}_\Sigma^r})$ as follows: For every $b \in \Sigma'$, each node labeled with b nondeterministically chooses a new label $a \in \Sigma$, such that $h(a) = b$. Then, the automaton A is simulated on that new input. ■

Proposition 3.9 ($[\![\mathrm{NLDA_g}]\!]_{\mathrm{DG}_\Sigma^r} \subset [\![\mathrm{ALDA_g}]\!]_{\mathrm{DG}_\Sigma^r}$).
▶ There are (infinitely many) $\mathrm{ALDA_g}$-recognizable digraph languages that are not $\mathrm{NLDA_g}$-recognizable. ◀

Proof. For any constant $k \geqslant 1$, we consider the language $L^{\mathrm{card}}_{\leqslant k}$ of all digraphs that have at most k nodes, i.e., $L^{\mathrm{card}}_{\leqslant k} = \big\{G \in \mathrm{DG} \;\big|\; |V^G| \leqslant k\big\}$. We can easily construct an

ALDA_g that recognizes this digraph language: In a universal branching, each node goes to $k+1$ different states in parallel. The automaton accepts if and only if there is no branch in which the $k+1$ states occur all at once. Now, assume for sake of contradiction that $L^{\text{card}}_{\leqslant k}$ is also recognized by some NLDA_g A, and let G be a digraph with k nodes. We construct a variant G' of G with $k+1$ nodes by duplicating some node v, together with all of its incoming and outgoing edges. Observe that any accepting run of A on G can be extended to an accepting run on G', where the copy of v behaves exactly like v in every round. ■

Proposition 3.10 (Closure properties of $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$).
▶ The class $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$ of NLDA_g-recognizable digraph languages is effectively closed under union, intersection and projection, but *not* closed under complementation. ◀

Proof. For union and projection, we simply use the same constructions as for ALDA_g's (see Proposition 3.8).

Intersection can be handled by a product construction, similar to the one for finite automata on words. Given two NLDA_g's A_1 and A_2, we construct an NLDA_g $A_\otimes$ that operates on the Cartesian product of the state sets of A_1 and A_2. It simulates the two automata simultaneously and accepts if and only if both of them reach an accepting configuration.

To see that $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$ is not closed under complementation, we recall from the proof of Proposition 3.9 that for any $k \geqslant 1$, the language $L^{\text{card}}_{\leqslant k}$ of all digraphs that have at most k nodes is not NLDA_g-recognizable. However, complementing the ALDA_g given for $L^{\text{card}}_{\leqslant k}$ yields an NLDA_g that recognizes the complement language $L^{\text{card}}_{\geqslant k+1}$. ■

Proposition 3.11 ($[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma} \subset [\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$).
▶ There are (infinitely many) NLDA_g-recognizable digraph languages that are not DLDA_g-recognizable. ◀

Proof. Let $k \geqslant 2$. As mentioned in the proof of Proposition 3.10, the language $L^{\text{card}}_{\geqslant k}$ of all digraphs that have at least k nodes is NLDA_g-recognizable. To see that it is not DLDA_g-recognizable, consider (similarly to the proof of Proposition 3.9) a digraph G with $k-1$ nodes and a variant G' with k nodes obtained from G by duplicating some node v, together with all of its incoming and outgoing edges. Given any DLDA_g A, the determinism of A guarantees that v and its copy v' behave the same way in the (unique) run of A on G'. Hence, if that run is accepting, so is the run on G. ■

Proposition 3.12 (Closure properties of $[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma}$).
▶ The class $[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma}$ of DLDA_g-recognizable digraph languages is effectively closed under Boolean set operations, but *not* closed under projection. ◀

Proof. To complement a DLDA_g, we can simply complement its set of accepting sets. The product construction for intersection of NLDA_g's mentioned in Proposition 3.10 remains applicable when restricted to DLDA_g's.

Closure under node projection does not hold because we can, for instance, construct a DLDA_g that recognizes the language $L^{\text{occur}}_{a,b,c}$ of all $\{a,b,c\}$-labeled digraphs in which each of the three node labels occurs at least once. However, projection under the mapping $h\colon \{a,b,c\} \to \{\varepsilon\}$, with $h(a) = h(b) = h(c) = \varepsilon$ (the empty word), yields the digraph language $h(L^{\text{occur}}_{a,b,c}) = L^{\text{card}}_{\geqslant 3}$, which is not DLDA_g-recognizable (see the proof of Proposition 3.11). ■

3.4 Equivalence with monadic second-order logic

Theorem 3.13 ($[\![\text{ALDA}_g]\!]_{\text{DG}^r_\Sigma} = [\![\text{MSOL}]\!]_{\text{DG}^r_\Sigma}$).
▶ A digraph language is ALDA_g-recognizable if and only if it is MSOL-definable. There are effective translations in both directions. ◀

This characterization is heavily inspired by the work of Löding and Thomas in [LT00].

Proof sketch. ($\Rightarrow$) We start with the direction $[\![\text{ALDA}_g]\!]_{\text{DG}^r_\Sigma} \subseteq [\![\text{MSOL}]\!]_{\text{DG}^r_\Sigma}$. Let $\mathcal{A} = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ be an ALDA_g of length n over Σ-labeled, r-relational digraphs. Without loss of generality, we may assume that every configuration reachable by $\mathcal{A}$ has at least one successor configuration and that no permanent configuration is reachable in less than n rounds. In order to encode the acceptance behavior of $\mathcal{A}$ into an MSOL-formula $\varphi_{\mathcal{A}}$, we use again the game-theoretic characterization briefly mentioned in the proof sketch of Proposition 3.8. Given $\mathcal{A}$ and some $G \in \text{DG}^r_\Sigma$, we consider a game with two players: the *automaton* (player $\exists$) and the *pathfinder* (player $\forall$). This game is represented by an acyclic digraph whose nodes are precisely the configurations reachable by $\mathcal{A}$ on G. For any two *nonpermanent* configurations $G[\zeta]$ and $G[\eta]$, there is a directed edge from $G[\zeta]$ to $G[\eta]$ if and only if $G[\eta] \in \delta_g(G[\zeta])$. Starting at the initial configuration $G[\delta_0 \circ \lambda^G]$, the two players move through the game together by following directed edges. If the current configuration is existential, then the automaton has to choose the next move, if it is universal, then the decision belongs to the pathfinder. This continues until some permanent configuration is reached. The automaton wins if that permanent configuration is accepting, whereas the pathfinder wins if it is rejecting. A player is said to have a *winning strategy* if it can always win, independently of its opponent's moves. It is straightforward to prove that the automaton has a winning strategy if and only if $\mathcal{A}$ accepts G. Our MSOL-formula $\varphi_{\mathcal{A}}$ will express the existence of such a winning strategy, and thus be equivalent to $\mathcal{A}$.

Within MSOL, we represent a path $G[\zeta_0]\cdots G[\zeta_n]$ through the game by a sequence of families of set variables $\vec{X}_0, \dots, \vec{X}_n$, where $\vec{X}_0 = (\,)$ and $\vec{X}_i = (X_{i,q})_{q \in Q}$, for $1 \leqslant i \leqslant n$. The intention is that each set variable $X_{i,q}$ is interpreted as the set of nodes $v \in V^G$ for which $\zeta_i(v) = q$. (We do not need set variables to represent $G[\zeta_0]$, since the players always start at $G[\delta_0 \circ \lambda^G]$.)

Now, for every round i, we construct a formula $\varphi_i^{\text{win}}(\vec{X}_i)$ (i.e., with free variables in $\vec{X}_i$), which expresses that the automaton has a winning strategy in the subgame starting at the configuration $G[\zeta_i]$ represented by $\vec{X}_i$. In case $G[\zeta_i]$ is existential, this is true if the automaton has a winning strategy in some successor configuration of $G[\zeta_i]$, whereas if $G[\zeta_i]$ is universal, the automaton must have a winning strategy in all successor configurations of $G[\zeta_i]$. This yields the following recursive definition for $0 \leqslant i \leqslant n-1$:

$$\varphi_i^{\text{win}}(\vec{X}_i) \coloneqq \exists_{\vec{X}_{i+1}} \Big(\varphi_{i+1}^{\text{succ}}(\vec{X}_i, \vec{X}_{i+1}) \wedge \varphi_{i+1}^{\text{win}}(\vec{X}_{i+1}) \Big),$$

if level i of $\mathcal{A}$ is existential, and

$$\varphi_i^{\text{win}}(\vec{X}_i) \coloneqq \forall_{\vec{X}_{i+1}} \Big(\varphi_{i+1}^{\text{succ}}(\vec{X}_i, \vec{X}_{i+1}) \rightarrow \varphi_{i+1}^{\text{win}}(\vec{X}_{i+1}) \Big),$$

if level i of $\mathcal{A}$ is universal. Here, $\varphi_{i+1}^{\text{succ}}(\vec{X}_i, \vec{X}_{i+1})$ is an FOL-formula expressing that $\vec{X}_i$ and $\vec{X}_{i+1}$ represent two configurations $G[\zeta_i]$ and $G[\zeta_{i+1}]$ such that $G[\zeta_{i+1}] \in \delta_g(G[\zeta_i])$. As our recursion base, we can easily construct a formula $\varphi_n^{\text{win}}(\vec{X}_n)$ that is satisfied if and only if $\vec{X}_n$ represents an accepting configuration of $\mathcal{A}$.

The desired MSOL-formula is $\varphi_{\mathcal{A}} \coloneqq \varphi_0^{\text{win}}(\vec{X}_0) = \varphi_0^{\text{win}}(\,)$.

(⇐) For the direction $[\![\text{ALDA}_g]\!]_{\text{DG}^r_\Sigma} \supseteq [\![\text{MSOL}]\!]_{\text{DG}^r_\Sigma}$, we can proceed by induction on the structure of an MSOL-formula φ. In order to deal with free occurrences of node symbols, we encode their interpretations into the node labels (which normally encode only the interpretations of set symbols). Let $\text{free}_0(\varphi)$ be an abbreviation for $\text{free}(\varphi) \cap \mathbb{S}_0$. For $G \in \text{DG}^r_\Sigma$ and $\alpha\colon \text{free}_0(\varphi) \to V^G$, we represent $G[\alpha]$ as the labeled digraph $G[\lambda^G \times \alpha^{-1}]$ whose labeling $\lambda^G \times \alpha^{-1}$ assigns to each node $v \in V^G$ the tuple $\big(\lambda^G(v),\ \alpha^{-1}(v)\big)$, where $\alpha^{-1}(v)$ is the set of all node symbols in $\text{free}_0(\varphi)$ to which α assigns v. We now inductively construct an ALDA_g $A_\varphi = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over r-relational digraphs labeled with the alphabet $\Sigma' = \Sigma \times 2^{\text{free}_0(\varphi)}$, such that

$$G[\lambda^G \times \alpha^{-1}] \in [\![A_\varphi]\!]_{\text{DG}^r_\Sigma}, \quad \text{if and only if} \quad G[\alpha] \models \varphi.$$

Base case. Let $x, y \in \mathbb{S}_0$, $X \in \mathbb{S}_1$ and $i \in [r]$. If φ is one of the atomic formulas $x \doteq y$ or $X(x)$, then, in A_φ, each node simply checks that its own label $(a, M) \in \Sigma \times 2^{\text{free}_0(\varphi)}$ satisfies the condition specified in φ (which, in particular, is the case if $x, y \notin M$). Since this can be directly encoded into the initialization function δ_0, the ALDA_g has length 0. It accepts the input digraph if and only if every node reports that its label satisfies the condition.

The case $\varphi = R_i(x, y)$ is very similar, but A_φ needs one communication round, after which the node assigned to y can check whether it has received a message through an i-edge from the node assigned to x. Accordingly, A_φ has length 1.

Inductive step. In case φ is a composed formula, we can obtain A_φ by means of the constructions outlined in the proof sketch of Proposition 3.8 (closure properties of $[\![\text{ALDA}_g]\!]_{\text{DG}^r_\Sigma}$). Let ψ and ψ' be MSOL-formulas with equivalent ALDA_g's A_ψ and $A_{\psi'}$, respectively.

If $\varphi = \neg\psi$, it suffices to define $A_\varphi = \overline{A}_\psi$. Similarly, if $\varphi = \psi \vee \psi'$, we get A_φ by applying the union construction on A_ψ and $A_{\psi'}$. (In general, we first have to extend A_ψ and $A_{\psi'}$ such that they both operate on the same node alphabet $\Sigma \times 2^{\text{free}_0(\psi) \cup \text{free}_0(\psi')}$.)

Existential quantification can be handled by node projection. If $\varphi = \exists_X(\psi)$, with $X \in \mathbb{S}_1$, we construct A_φ by applying the projection construction on A_ψ, using a mapping $h\colon \Sigma \times 2^{\text{free}_0(\psi)} \to \tilde{\Sigma} \times 2^{\text{free}_0(\psi)}$ that deletes the set variable X from every label. In other words, the new alphabet $\tilde{\Sigma}$ encodes the subsets of $\text{free}(\psi) \cap (\mathbb{S}_1 \setminus \{X\})$. An analogous approach can be used if $\varphi = \exists_x(\psi)$, with $x \in \mathbb{S}_0$. The only difference is that, instead of applying the projection construction directly on A_ψ, we apply it on a variant A'_ψ that operates just like A_ψ, but additionally checks that precisely one node in the input digraph is assigned to the node variable x. ■

From Theorem 3.13 we can immediately infer that it is undecidable whether the digraph language recognized by some arbitrary ALDA_g is empty. Otherwise, we could decide the satisfiability problem of MSOL on digraphs, which is known to be undecidable (a direct consequence of Trakhtenbrot's Theorem, see, e.g., [Lib04, Thm 9.2]).

Corollary 3.14 (Emptiness problem of ALDA_g's).
▶ The emptiness problem for ALDA_g's is undecidable. ◀

3.5 Emptiness problem for nondeterministic automata

At the cost of reduced expressive power, we can also obtain a positive decidability result.

Proposition 3.15 (Emptiness problem of NLDA$_g$'s).
▶ The emptiness problem of NLDA$_g$'s is decidable in doubly-exponential time. More precisely, for any NLDA$_g$ $A = (\vec{Q}, \delta_0, \delta, \mathcal{F})$ over Σ-labeled, r-relational digraphs, whether its recognized digraph language $[\![A]\!]_{\mathrm{DG}^r_\Sigma}$ is empty or not can be decided in time 2^k, where $k \in O(r \cdot |Q|^{4\,\mathrm{len}(A)} \cdot \mathrm{len}(A))$.

Furthermore, whether or not the digraph language $[\![A]\!]_{\mathrm{DG}^r_\Sigma}$ contains any *connected*, *undirected* graph can be decided in time $2^{2^{k'}}$, where $k' \in O(r \cdot |Q| \cdot \mathrm{len}(A))$. ◀

Proof sketch. Let $G \in \mathrm{DG}^r_\Sigma$. Since NLDA$_g$'s cannot perform universal branching, we can consider any run of A on G as a sequence of configurations $\rho = G[\zeta_0]\cdots G[\zeta_n]$, with $n \leqslant \mathrm{len}(A)$. In ρ, each node of G traverses one of at most $|Q|^{\mathrm{len}(A)+1}$ possible sequences of states. Now, assume that G has more than $|Q|^{\mathrm{len}(A)+1}$ nodes. Then, by the Pigeonhole Principle, there must be two distinct nodes $v, v' \in V^G$ that traverse the same sequence of states in ρ. We construct a smaller digraph G' by removing v' from G, together with its adjacent edges, and adding directed edges from v to all of the former *outgoing* neighbors of v'. If all the nodes in G' maintain their nondeterministic choices from ρ, none of them will notice that v' is missing, and consequently they all behave just as in ρ. The resulting run ρ' on G' is accepting if and only if ρ is accepting.

Applying this argument recursively, we conclude that if $[\![A]\!]_{\mathrm{DG}^r_\Sigma}$ is not empty, then it must contain some labeled digraph that has at most $|Q|^{\mathrm{len}(A)+1}$ nodes. Hence, the emptiness problem is decidable because the search space is finite. The time complexity indicated above corresponds to the naive approach of checking every digraph with at most $|Q|^{\mathrm{len}(A)+1}$ nodes.

If we are only interested in connected, undirected graphs, the reasoning is very similar, but we have to require a larger minimum number of nodes in order to be able to remove some node without influencing the behavior of the others. In a graph G with more than $\left(|Q| \cdot 2^{r \cdot |Q|}\right)^{\mathrm{len}(A)+1}$ nodes, there must be two distinct nodes $v, v' \in V^G$ that, in addition to traversing the same sequence of states, also receive the same family of sets of states from their neighborhood in every round. Observe that the automaton will not notice if we merge v and v'. The rest of the argument is analogous to the previous scenario. ∎

3.6 Summary and discussion

We have introduced ALDA$_g$'s, which are probably the first graph automata in the literature to be equivalent to MSOL on digraphs. However, their expressive power results mainly from the use of alternation: we have seen that the deterministic, nondeterministic and alternating variants form a strict hierarchy, i.e.,

$$[\![\mathrm{DLDA_g}]\!]_{\mathrm{DG}^r_\Sigma} \subset [\![\mathrm{NLDA_g}]\!]_{\mathrm{DG}^r_\Sigma} \subset [\![\mathrm{ALDA_g}]\!]_{\mathrm{DG}^r_\Sigma}.$$

The corresponding closure and decidability properties are summarized in Table 3.1.

	Closure Properties				Decidability
	Complement	Union	Intersection	Projection	Emptiness
ALDA$_g$	✓	✓	✓	✓	✗
NLDA$_g$	✗	✓	✓	✓	✓
DLDA$_g$	✓	✓	✓	✗	✓

Table 3.1. Closure and decidability properties of alternating, nondeterministic, and deterministic LDA$_g$'s.

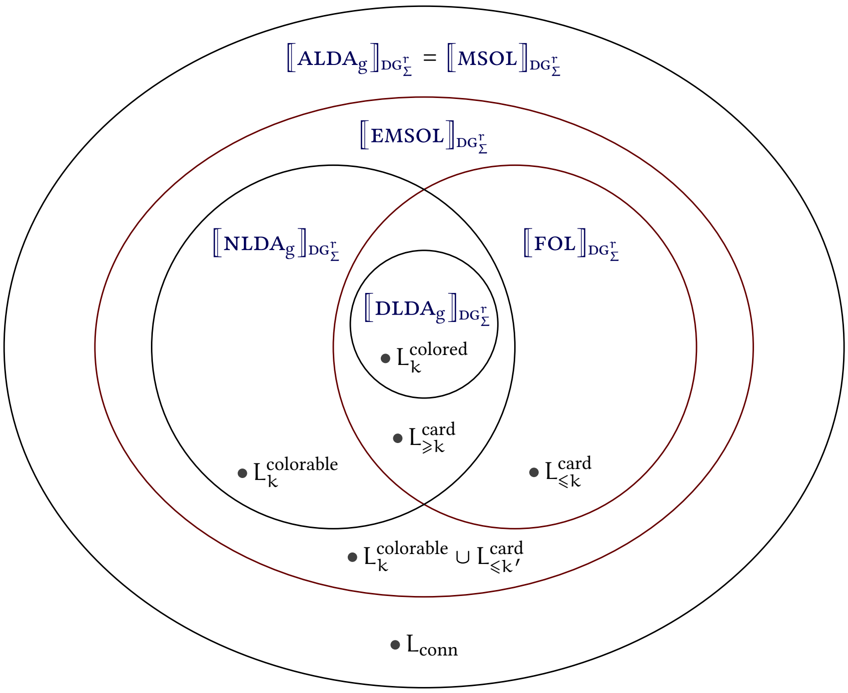

Figure 3.6. Venn diagram relating the classes of digraph languages recognizable by our three flavors of LDA$_g$'s to those definable in MSOL, EMSOL and FOL.

On an intuitive level, this hierarchy and these closure properties do not seem very surprising. One might even ask: *are* ALDA$_g$*'s just another syntax for* MSOL*?* Indeed, universal branchings correspond to universal quantification, and nondeterministic choices to existential quantification. By disallowing universal set quantification in MSOL we obtain EMSOL, and further disallowing existential set quantification yields FOL. Analogously to LDA$_g$'s, the classes of digraph languages definable in these logics form a strict hierarchy, i.e.,

$$⟦\text{FOL}⟧_{\text{DG}^r_\Sigma} \subset ⟦\text{EMSOL}⟧_{\text{DG}^r_\Sigma} \subset ⟦\text{MSOL}⟧_{\text{DG}^r_\Sigma}.$$

Furthermore, the closure properties of $⟦\text{EMSOL}⟧_{\text{DG}^r_\Sigma}$ and $⟦\text{FOL}⟧_{\text{DG}^r_\Sigma}$ coincide with those of $⟦\text{NLDA}_g⟧_{\text{DG}^r_\Sigma}$ and $⟦\text{DLDA}_g⟧_{\text{DG}^r_\Sigma}$, respectively. Given that $⟦\text{ALDA}_g⟧_{\text{DG}^r_\Sigma}$ and $⟦\text{MSOL}⟧_{\text{DG}^r_\Sigma}$ are equal, one might therefore expect that the analogous equalities hold for the weaker classes. However, as already hinted by the positive decidability properties in Table 3.1, this is not the case. The actual relationships between the different classes of digraph languages are depicted in Figure 3.6. A glance at this diagram suggests that ALDA_g's are not simply a one-to-one reproduction of MSOL.

Justification of Figure 3.6. Fagin has shown in [Fag75] that the language L_{conn} of all (weakly) connected digraphs separates $[\![\text{EMSOL}]\!]_{\text{DG}^r_\Sigma}$ from $[\![\text{MSOL}]\!]_{\text{DG}^r_\Sigma}$. (Since non-connectivity is EMSOL-definable, this also implies that $[\![\text{EMSOL}]\!]_{\text{DG}^r_\Sigma}$ is not closed under complementation.) The inclusion $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma} \subseteq [\![\text{EMSOL}]\!]_{\text{DG}^r_\Sigma}$ holds because we can encode every NLDA_g into an EMSOL-formula, using the same construction as in the proof sketch of Theorem 3.13. It is also easy to see that we can encode a DLDA_g by inductively constructing a family of FOL-formulas $\varphi^{\text{state}}_{i:q}(x)$, stating that in round i (of the *unique* run of the automaton), the node assigned to x is in state q. Hence, $[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma} \subseteq [\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$. In the following, let $k, k' \geqslant 2$. The incomparability of $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$ and $[\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$ is witnessed by the language $L^{\text{colorable}}_k$ of k-colorable digraphs, which lies within $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$ (see Example 3.1) but outside of $[\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$ (see, e.g., [Lib04]), and the language $L^{\text{card}}_{\leqslant k}$ of digraphs with at most k nodes, which lies outside of $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma}$ (see the proof of Proposition 3.9) but obviously within $[\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$. Considering the union language $L^{\text{colorable}}_k \cup L^{\text{card}}_{\leqslant k'}$ also tells us that the inclusion of $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma} \cup [\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$ in $[\![\text{EMSOL}]\!]_{\text{DG}^r_\Sigma}$ is strict. Finally, the language $L^{\text{card}}_{\geqslant k}$ of digraphs with at least k nodes separates $[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma}$ from $[\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma} \cap [\![\text{FOL}]\!]_{\text{DG}^r_\Sigma}$ (see the proof of Proposition 3.11). A simple example of a language that lies within $[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma}$ is the set L^{colored}_k of Σ-labeled digraphs whose labelings are valid k-colorings, with $|\Sigma| = k$. ∎

Nevertheless, based on the equivalence of LDA's and $\overleftarrow{\text{ML}}$ established by Hella et al. (Theorem 2.5), we can actually obtain precise logical characterizations of DLDA_g's and NLDA_g's by extending $\overleftarrow{\text{ML}}$ with global modalities and existential set quantifiers. Adapting the proofs of [HJK+12, HJK+15] to our setting, it is relatively easy to show that

$$[\![\text{DLDA}_g]\!]_{\text{DG}^r_\Sigma} = [\![\overleftarrow{\text{ML}}_g]\!]_{\text{DG}^r_\Sigma} \quad \text{and} \quad [\![\text{NLDA}_g]\!]_{\text{DG}^r_\Sigma} = [\![\Sigma^{\text{MSO}}_1(\overleftarrow{\text{ML}}_g)]\!]_{\text{DG}^r_\Sigma}.$$

More generally, one can show a levelwise equivalence with the set quantifier alternation hierarchy of $\text{MSO}(\overleftarrow{\text{ML}}_g)$, a rather unconventional logic that is equivalent to MSOL. In other words, two corresponding levels of alternation in the frameworks of ALDA_g's and $\text{MSO}(\overleftarrow{\text{ML}}_g)$ characterize exactly the same digraph languages. Against this backdrop, ALDA_g's may be best described as a machine-oriented syntax for $\text{MSO}(\overleftarrow{\text{ML}}_g)$. We shall pick up on this point in the introduction of Chapter 6.

As of the time of writing this thesis, no new results on $[\![\text{MSOL}]\!]_{\text{DG}^r_\Sigma}$ have been inferred from the alternative characterization through ALDA_g's. On the other hand, the notion of NLDA_g contributes to the general observation, mentioned in Section 1.1.1, that many characterizations of regularity, which are equivalent on words and trees, drift apart on digraphs. To see this, consider NLDA_g's whose input is restricted to those Σ-labeled, r-relational digraphs that represent words or trees over the alphabet Σ. For words, $r = 1$ and edges simply go from one position to the next, whereas for ordered trees of arity k, we set $r = k$ and require edge relations such that $uv \in R^G_i$ if and only if u is the i-th child of v. Observe that we can easily simulate any word or tree automaton by an NLDA_g of length 2: guess a run of the automaton in the first round (each node nondeterministically chooses some state), then check whether it is a valid accepting run in the second round (transitions are verified locally, and acceptance is determined by the unique sink). This implies that the classes of NLDA_g-recognizable and MSOL-definable languages collapse on words and trees, and hence that $\mathrm{NLDA_g}$'s recognize precisely the regular languages on those restricted structures. (Note that this does not hold for $\mathrm{DLDA_g}$'s; for instance, it is easy to see that they cannot decide whether a given unary word is of even length.) In this light, the decidability of the emptiness problem for $\mathrm{NLDA_g}$'s can be seen as an extension to arbitrary digraphs of the corresponding decidability results for finite automata on words and trees.

Chapter based on the conference paper [Rei17].

4 Asynchronous Nonlocal Automata

In this chapter, we introduce a particular class of nonlocal distributed automata and show that on finite digraphs, they are equivalent to the least-fixpoint fragment of the backward μ-calculus, or simply *backward* μ*-fragment*.

For the general case, a logical characterization has been provided by Kuusisto in [Kuu13a]; there he introduced a modal-logic-based variant of Datalog, called *modal substitution calculus*, that captures exactly the class of nonlocal automata. Furthermore, [Kuu13a, Prp. 7] shows that these automata can easily recognize all the properties definable in the backward μ-fragment on finite digraphs. On the other hand, the reverse conversion from nonlocal automata to the backward μ-fragment is not possible in general. As explained in [Kuu13a, Prp. 6], it is easy to come up with an automaton that makes crucial use of the fact that a node can determine whether it receives the same information from all of its incoming neighbors at exactly the same time. Such synchronous behavior cannot be simulated in the backward μ-fragment (and not even in MSOL). This leaves open the problem of identifying a subclass of distributed automata for which the conversion works in both directions.

Here, we present a very simple solution: it basically suffices to transfer the standard notion of asynchronous algorithm to the setting of distributed automata.

The organisation of this chapter is as follows. After giving the necessary formal definitions in Section 4.1, we state and briefly discuss the main result in Section 4.2. The proof is then developed in the last two sections. Section 4.3 presents the rather straightforward translation from logic to automata. The reverse translation is given in Section 4.4, which is a bit more involved and therefore occupies the largest part of the chapter.

4.1 Preliminaries

The class of asynchronous distributed automata introduced in this chapter, is a special case of the distributed automata defined in Section 2.7. We maintain the same syntax as in Definition 2.3, but reintroduce the semantics of (unrestricted) distributed automata from a slightly different perspective. In order to keep notation simple, we do this only for 1-relational digraphs, but everything presented here can easily be extended to the multi-relational case.

To run a distributed automaton A on a digraph G, we now regard the edges of G as FIFO buffers. Each buffer vw will always contain a sequence of states previously traversed by node v. An adversary chooses when v evaluates δ to push a new state to the back of the buffer, and when the current first state gets popped from the front. The details are clarified in the following.

A *trace* of an automaton $A = (Q, \delta_0, \delta, F)$ is a finite nonempty sequence $\sigma = q_1 \dots q_n$ of states in Q such that $q_i \neq q_{i+1}$ and $\delta(q_i, S_i) = q_{i+1}$ for some $S_i \subseteq Q$. Notice that A is quasi-acyclic if and only if its set of traces $\mathfrak{Q}$ is finite.

For any states $p, q \in Q$ and any (possibly empty) sequence σ of states in Q, we define the unary postfix operators first, last, pushlast and popfirst as follows:

$$p\sigma.\mathsf{first} = \sigma p.\mathsf{last} = p,$$

$$\sigma p.\mathsf{pushlast}(q) = \begin{cases} \sigma p q & \text{if } p \neq q, \\ \sigma p & \text{if } p = q, \end{cases}$$

$$p\sigma.\mathsf{popfirst} = \begin{cases} \sigma & \text{if } \sigma \text{ is nonempty}, \\ p\sigma & \text{if } \sigma \text{ is empty}. \end{cases}$$

An (asynchronous) *timing* of a digraph $G = (V^G, R^G, \lambda^G)$ is an infinite sequence $\tau = (\tau_1, \tau_2, \tau_3, \dots)$ of maps $\tau_t \colon V^G \cup R^G \to \mathbb{2}$, indicating which nodes and edges are active at time t, where 1 is assigned infinitely often to every node and every edge. More formally, for all $t \in \mathbb{N}_+$, $v \in V^G$ and $e \in R^G$, there exist $i, j > t$ such that $\tau_i(v) = 1$ and $\tau_j(e) = 1$. We refer to this as the *fairness property* of τ. As a restriction, we say that τ is *lossless-asynchronous* if $\tau_t(uv) = 1$ implies $\tau_t(v) = 1$ for all $t \in \mathbb{N}_+$ and $uv \in R^G$. Furthermore, τ is called the (unique) *synchronous timing* of G if $\tau_t(v) = \tau_t(e) = 1$ for all $t \in \mathbb{N}_+$, $v \in V^G$ and $e \in R^G$.

Definition 4.1 (Asynchronous Run).

▶ Let $A = (Q, \delta_0, \delta, F)$ be a distributed automaton over s-bit labeled digraphs and $\mathfrak{Q}$ be its set of traces. Furthermore, let $G = (V^G, R^G, \lambda^G)$ be an s-bit labeled digraph and $\tau = (\tau_1, \tau_2, \tau_3, \dots)$ be a timing of G. The (asynchronous) *run* of A on G timed by τ is the infinite sequence $\rho = (\rho_0, \rho_1, \rho_2, \dots)$ of *configurations* $\rho_t \colon V^G \cup R^G \to \mathfrak{Q}$, with $\rho_t(V^G) \subseteq Q$, which are defined inductively as follows, for $t \in \mathbb{N}$, $v \in V^G$ and $vw \in R^G$:

$$\rho_0(v) = \rho_0(vw) = \delta_0(\lambda^G(v)),$$

$$\rho_{t+1}(v) = \begin{cases} \rho_t(v) & \text{if } \tau_{t+1}(v) = 0, \\ \delta\big(\rho_t(v), \{\rho_t(uv).\mathsf{first} \mid uv \in R^G\}\big) & \text{if } \tau_{t+1}(v) = 1, \end{cases}$$

$$\rho_{t+1}(vw) = \begin{cases} \rho_t(vw).\mathsf{pushlast}(\rho_{t+1}(v)) & \text{if } \tau_{t+1}(vw) = 0, \\ \rho_t(vw).\mathsf{pushlast}(\rho_{t+1}(v)).\mathsf{popfirst} & \text{if } \tau_{t+1}(vw) = 1. \end{cases}$$

If τ is the synchronous timing of G, we refer to ρ as the *synchronous run* of A on G. ◀

Throughout this chapter, we assume that our digraphs, automata and logical formulas agree on the number s of labeling bits. An automaton A *accepts* a pointed digraph $G[v]$ under timing τ if v visits an accepting state at some point in the run ρ

of A on G timed by τ, i.e., if there exists $t \in \mathbb{N}$ such that $\rho_t(v) \in F$. If we simply say that A accepts $G[v]$, without explicitly specifying a timing τ, then we stipulate that ρ is the synchronous run of A on G. Notice that this is coherent with the definition of acceptance presented in Section 2.7.

Given a digraph $G = (V^G, R^G, \lambda^G)$ and a class T of timings of G, the automaton A is called *consistent* for G and T if for all $v \in V^G$, either A accepts $G[v]$ under every timing in T, or A does not accept $G[v]$ under any timing in T. We say that A is *asynchronous* if it is consistent for every possible choice of G and T, and *lossless-asynchronous* if it is consistent for every choice where T contains only lossless-asynchronous timings. By contrast, we call an automaton *synchronous* if we wish to emphasize that no such consistency requirements are imposed. Intuitively, all automata can operate in the synchronous setting, but only some of them also work reliably in environments that provide fewer guarantees.

We denote by a-DA, la-DA and DA the classes of asynchronous, lossless-asynchronous and synchronous automata, respectively. Similarly, a-QDA, la-QDA and QDA are the corresponding classes of quasi-acyclic automata.

Next, we want to introduce the backward μ-fragment, for which it is convenient to distinguish explicitly between constants and variables. As our starting point, we consider $\overleftarrow{\mathrm{ML}}$ restricted to s set constants and (arbitrarily many) unnegated set variables. Its formulas are generated by the grammar

$$\varphi ::= \bot \mid \top \mid P_i \mid \neg P_i \mid X \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid \overline{\Diamond}\varphi \mid \overline{\Box}\varphi\,,$$

where $P_i \in \mathbb{S}_1$ is considered to be a set constant, for $1 \leqslant i \leqslant s$, and $X \in \mathbb{S}_1 \setminus \{P_1, \dots, P_s\}$ is considered to be a set variable. Note that this syntax ensures that set variables cannot be negated.

Traditionally, the modal μ-calculus is defined to comprise individual fixpoints which may be nested. However, it is well-known that we can add simultaneous fixpoints to the μ-calculus without changing its expressive power, and that nested fixpoints of the same type (i.e., least or greatest) can be rewritten as non-nested simultaneous ones (see, e.g., [BS07, § 3.7] or [Len05, § 4.3]). The following definition directly takes advantage of this fact. We shall restrict ourselves to the *μ-fragment of the backward μ-calculus*, abbreviated *backward μ-fragment*, where only least fixpoints are allowed, and where the usual modal operators are replaced by their backward-looking variants. Without loss of generality, we stipulate that each formula of the backward μ-fragment with s set constants is of the form

$$\varphi \;=\; \mu\begin{pmatrix} X_1 \\ \vdots \\ X_k \end{pmatrix}.\begin{pmatrix} \varphi_1(P_1, \dots, P_s, X_1, \dots, X_k) \\ \vdots \\ \varphi_k(P_1, \dots, P_s, X_1, \dots, X_k) \end{pmatrix},$$

where $X_1, \dots, X_k \in \mathbb{S}_1 \setminus \{P_1, \dots, P_s\}$ are considered to be set variables, and $\varphi_1, \dots, \varphi_k$ are formulas of $\overleftarrow{\mathrm{ML}}$ with s set constants and unnegated set variables that may contain no other set variables than $X_1, \dots, X_k$. We shall denote the set of formulas of the backward μ-fragment by $\Sigma_1^\mu(\overleftarrow{\mathrm{ML}})$.

For every digraph $G = (V^G, R^G, \lambda^G)$, the tuple $(\varphi_1, \dots, \varphi_k)$ gives rise to an operator $f\colon (2^{V^G})^k \to (2^{V^G})^k$ that takes some valuation of $\vec{X} = (X_1, \dots, X_k)$ and reassigns to each X_i the resulting valuation of φ_i. More formally, f maps $\vec{W} = (W_1, \dots, W_k)$ to $(W_1', \dots, W_k')$ such that $W_i' = [\![\varphi_i]\!]_{G[\vec{X} \mapsto \vec{W}]}$. Here, $G[\vec{X} \mapsto \vec{W}]$ is the extended

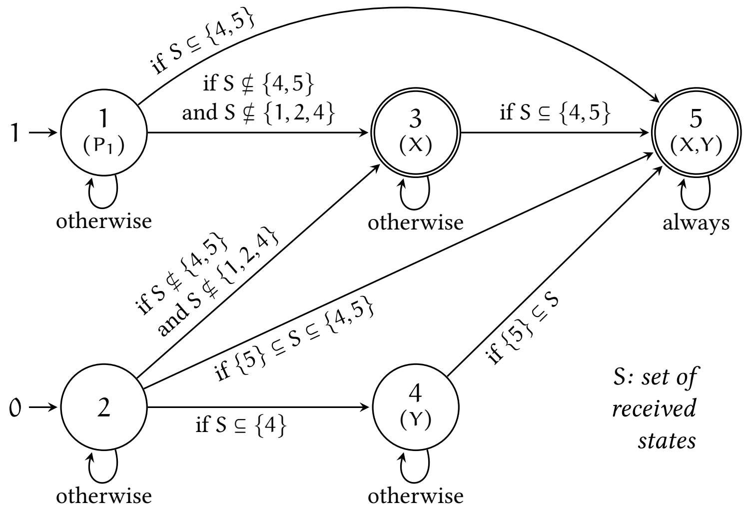

Figure 4.1. A quasi-acyclic asynchronous distributed automaton that is equivalent to the formula

$$\mu \binom{X}{Y} . \begin{pmatrix} (P_1 \wedge Y) \vee \overleftarrow{\Diamond} X \\ \overleftarrow{\Box}\, Y \end{pmatrix}$$

of the backward μ-fragment. A given 1-bit labeled pointed digraph $G[v]$ is accepted by this automaton if and only if, starting at v and following G's edges in the backward direction, it is possible to reach some node u labeled with 1 from which it is impossible to reach any directed cycle.

variant of G that interprets each X_i as W_i. A (simultaneous) *fixpoint* of the operator f is a tuple $\vec{W} \in (2^{V^G})^k$ such that $f(\vec{W}) = \vec{W}$. Since, by definition, set variables occur only positively in formulas, the operator f is *monotonic*. This means that $\vec{W} \subseteq \vec{W}'$ implies $f(\vec{W}) \subseteq f(\vec{W}')$ for all $\vec{W}, \vec{W}' \in (2^{V^G})^k$, where set inclusions are to be understood componentwise (i.e., $W_i \subseteq W_i'$ for each i). Therefore, by virtue of a theorem due to Knaster and Tarski, f has a *least fixpoint*, which is defined as the unique fixpoint $\vec{U} = (U_1, \dots, U_k)$ of f such that $\vec{U} \subseteq \vec{W}$ for every other fixpoint $\vec{W}$ of f. As a matter of fact, the Knaster-Tarski theorem even tells us that $\vec{U}$ is equal to $\bigcap\{\vec{W} \in (2^{V^G})^k \mid f(\vec{W}) \subseteq \vec{W}\}$, where set operations must also be understood componentwise. Another, perhaps more intuitive, way of characterizing $\vec{U}$ is to consider the inductively constructed sequence of approximants $(\vec{U}^0, \vec{U}^1, \vec{U}^2, \dots)$, where $\vec{U}^0 = (\varnothing, \dots, \varnothing)$ and $\vec{U}^{j+1} = f(\vec{U}^j)$. Since this sequence is monotonically increasing and V^G is finite, there exists $n \in \mathbb{N}$ such that $\vec{U}^n = \vec{U}^{n+1}$. It is easy to check that $\vec{U}^n$ coincides with the least fixpoint $\vec{U}$. For more details and proofs, see, e.g., [GKL+07, § 3.3.1].

Having introduced the necessary background, we can finally establish the semantics of φ with respect to G: the set $[\![\varphi]\!]_G = \{v \in V^G \mid G[v] \models \varphi\}$ of nodes at which φ holds is precisely U_1, the first component of $\vec{U}$. Accordingly, the pointed digraph $G[v]$ lies in the language $[\![\varphi]\!]_{@\mathrm{DG}_s^1}$ defined by φ if and only if $v \in U_1$, and we denote by $[\![\Sigma_1^\mu(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}_s^1}$ the class of all pointed-digraph languages defined by some formula of the backward μ-fragment.

Figure 4.1 provides an example of a quasi-acyclic asynchronous distributed automaton and an equivalent formula of the backward μ-fragment.

$$\begin{tikzcd}
{[\![\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} & {[\![\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] \\
{[\![\text{la-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[u] & {[\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] \arrow[u] \\
{[\![\text{a-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[u] & {[\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] \arrow[u]
\end{tikzcd}$$

a. Immediate by the definitions.

$$\begin{tikzcd}
{[\![\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} & {[\![\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] & \\
{[\![\text{la-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[u] & {[\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] \arrow[u] \arrow[dr] & \\
 & & {[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[dl] \\
{[\![\text{a-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[uu] & {[\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}} \arrow[l] \arrow[uu] &
\end{tikzcd}$$

b. Collapse shown in this chapter.

Figure 4.2. Hierarchy of the classes of pointed-digraph languages recognizable by distributed automata (DA), depending on whether the automata are synchronous (neither “la” nor “a”), lossless-asynchronous (“la”), asynchronous (“a”), or quasi-acyclic (“Q”). The arrows denote inclusion (e.g., $[\![\text{la-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}} \subseteq [\![\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}$).

4.2 Equivalence with the backward mu-fragment

Based on the definitions given in Section 4.1, asynchronous automata are a special case of lossless-asynchronous automata, which in turn are a special case of synchronous automata. Furthermore, quasi-acyclicity constitutes an additional (possibly orthogonal) restriction on these models. We thus immediately obtain the hierarchy of classes depicted in Figure 4.2a.

This may seem counterintuitive at first sight, but it is actually consistent with the standard terminology of distributed computing: an asynchronous algorithm can always serve as a synchronous algorithm (i.e., it can be executed in a synchronous environment), but the converse is not true.

Our main result provides a simplification of this hierarchy: the classes $[\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$ and $[\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$ are actually equal to the class of pointed-digraph languages definable in the backward μ-fragment. This yields the revised diagram shown in Figure 4.2b.

Theorem 4.2 ($[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}} = [\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}} = [\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$).
▸ When restricted to finite digraphs, the backward μ-fragment is effectively equivalent to the classes of quasi-acyclic asynchronous automata and quasi-acyclic lossless-asynchronous automata. ◂

Proof. The forward direction is given by Proposition 4.3 (in Section 4.3), which asserts that $[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}} \subseteq [\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$, and the trivial observation that $[\![\text{a-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}} \subseteq [\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$. For the backward direction, we use Proposition 4.6 (in Section 4.4), which asserts that $[\![\text{la-}\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}} \subseteq [\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}}$. ∎

As stated before, synchronous automata are more powerful than the backward μ-fragment (and incomparable with MSOL). This holds even if we consider only quasi-acyclic automata, i.e., the inclusion $[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}} \subset [\![\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}}$ is known to be strict (see [Kuu13a, Prp. 6]). Moreover, an upcoming paper will show that the inclusion $[\![\mathrm{QDA}]\!]_{@\mathrm{DG}^{!}_{s}} \subset [\![\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}$ is also strict.

In contrast, it remains open whether quasi-acyclicity is in fact necessary for characterizing $[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}}$. On the one hand, this notion is crucial for our proof (see Proposition 4.6), but on the other hand, no pointed-digraph language separating $[\![\text{a-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}$ or $[\![\text{la-}\mathrm{DA}]\!]_{@\mathrm{DG}^{!}_{s}}$ from $[\![\Sigma^{\mu}_{1}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}^{!}_{s}}$ has been found so far.

4.3 Computing least fixpoints using asynchronous automata

In this section, we prove the easy direction of the main result. Given a formula φ of the backward μ-fragment, it is straightforward to construct a (synchronous) distributed automaton A that computes on any digraph the least fixpoint $\vec{U}$ of the operator associated with φ. As long as it operates in the synchronous setting, A simply follows the sequence of approximants $(\vec{U}^0, \vec{U}^1, \dots)$ described in Section 4.1. It is important to stress that the very same observation has previously been made in [Kuu13a, Prp. 7] (formulated from a different point of view). In the following proposition, we refine this observation by giving a more precise characterization of the obtained automaton: it is always quasi-acyclic and capable of operating in a (possibly lossy) asynchronous environment.

Proposition 4.3 ($[\![\Sigma_1^{\mu}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}_s^1} \subseteq [\![\text{a-QDA}]\!]_{@\mathrm{DG}_s^1}$).
▶ For every formula of the backward μ-fragment, we can effectively construct an equivalent quasi-acyclic asynchronous automaton. ◀

Proof. Let $\varphi = \mu(X_1, \dots, X_k).(\varphi_1, \dots, \varphi_k)$ be a formula of the backward μ-fragment with s set constants. Without loss of generality, we may assume that the subformulas $\varphi_1, \dots, \varphi_k$ do not contain any nested modalities. To see this, suppose that $\varphi_i = \overline{\Diamond}\psi$. Then φ is equivalent to $\varphi' = \mu(X_1, \dots, X_i, \dots, X_k, Y).(\varphi_1, \dots, \varphi_i', \dots, \varphi_k, \psi)$, where Y is a fresh set variable and $\varphi_i' = \overline{\Diamond}Y$. The operator $\overline{\Box}$ and Boolean combinations of $\overline{\Diamond}$ and $\overline{\Box}$ are handled analogously.

We now convert φ into an equivalent automaton $A = (Q, \delta_0, \delta, F)$ with state set $Q = 2^{\{P_1, \dots, P_s, X_1, \dots, X_k\}}$. The idea is that each node v of the input digraph has to remember which of the atomic propositions $P_1, \dots, P_s, X_1, \dots, X_k$ have, so far, been verified to hold at v. Therefore, we define the initialization function such that $\delta_0(x) = \{P_i \mid x(i) = 1\}$ for all $x \in 2^s$. Let us write $(q, S) \models \varphi_i$ to indicate that a pair $(q, S) \in Q \times 2^Q$ satisfies a subformula φ_i of φ. This is the case precisely when φ_i holds at any node v that satisfies exactly the atomic propositions in q and whose incoming neighbors satisfy exactly the propositions specified by S. Note that this satisfaction relation is well-defined in our context because the nesting depth of modal operators in φ_i is at most 1. With that, the transition function of A can be succinctly described by $\delta(q, S) = q \cup \{X_i \mid (q, S) \models \varphi_i\}$. Since $q \subseteq \delta(q, S)$, we are guaranteed that the automaton is quasi-acyclic. Finally, the accepting set is given by $F = \{q \mid X_1 \in q\}$.

It remains to prove that A is asynchronous and equivalent to φ. For this purpose, let $G = (V^G, R^G, \lambda^G)$ be an s-bit labeled digraph and $\vec{U} = (U_1, \dots, U_k) \in (2^{V^G})^k$ be the least fixpoint of the operator f associated with $(\varphi_1, \dots, \varphi_k)$. Due to the asynchrony condition, we must consider an arbitrary timing $\tau = (\tau_1, \tau_2, \dots)$ of G. The corresponding run $\rho = (\rho_0, \rho_1, \dots)$ of A on G timed by τ engenders an infinite sequence $(\vec{W}^0, \vec{W}^1, \dots)$, where each tuple $\vec{W}^t = (W_1^t, \dots, W_k^t) \in (2^{V^G})^k$ specifies the valuation of every set variable X_i at time t, i.e., $W_i^t = \{v \in V^G \mid X_i \in \rho_t(v)\}$. Since A is quasi-acyclic and V^G is finite, this sequence must eventually stabilize at some value $\vec{W}^\infty$, and each node accepts if and only if it belongs to W_1^∞. Reformulated this way, our task is to demonstrate that $\vec{W}^\infty$ equals $\vec{U}$, regardless of the timing τ.

"$\vec{W}^\infty \subseteq \vec{U}$": We show by induction that $\vec{W}^t \subseteq \vec{U}$ for all $t \in \mathbb{N}$. This obviously holds for $t = 0$, since $\vec{W}^0 = (\varnothing, \dots, \varnothing)$. Now, consider any node $v \in V^G$ at an arbitrary time t. Let q be the current state of v and S be the set of current states of its incoming neighbors. Depending on τ, it might be the case that v actually receives some outdated information S' instead of S. However, given that the neighbors' previous states cannot contain more set variables than their current ones (by construction), and that set variables can only occur positively in each φ_i, we know that $(q, S') \models \varphi_i$ implies $(q, S) \models \varphi_i$. Hence, if v performs a local transition at time t, then the only new set variables that can be added to its state must lie in $\{X_i \mid (q, S) \models \varphi_i\}$. On a global scale, this means that $\vec{W}^{t+1} \setminus \vec{W}^t \subseteq f(\vec{W}^t)$. Furthermore, by the induction hypothesis, the monotonicity of f, and the fact that $\vec{U}$ is a fixpoint, we have $f(\vec{W}^t) \subseteq f(\vec{U}) = \vec{U}$. Putting both together, and again relying on the induction hypothesis, we obtain $\vec{W}^{t+1} \subseteq \vec{U}$.

"$\vec{W}^\infty \supseteq \vec{U}$": For the converse direction, we make use of the Knaster-Tarski theorem, which gives us the equality $\vec{U} = \bigcap\{\vec{W} \in (2^{V^G})^k \mid f(\vec{W}) \subseteq \vec{W}\}$. With this, it suffices to show that $f(\vec{W}^\infty) \subseteq \vec{W}^\infty$. Consider some time $t \in \mathbb{N}$ such that $\vec{W}^{t'} = \vec{W}^\infty$ for all $t' \geqslant t$. Although we know that every node has reached its final state at time t, the FIFO buffers of some edges might still contain obsolete states from previous times. However, the fairness property of τ guarantees that our customized popfirst operation is executed infinitely often at every edge, while the pushlast operation has no effect because all the states remain unchanged. Therefore, there must be a time $t' \geqslant t$ from which on each buffer contains only the current state of its incoming node, i.e., $\rho_{t''}(uv) = \rho_{t''}(u)$ for all $t'' \geqslant t'$ and $uv \in R^G$. Moreover, the fairness property of τ also ensures that every node v reevaluates the local transition function δ infinitely often, based on its own current state q and the set S of states in the buffers associated with its incoming neighbors. As this has no influence on v's state, we can deduce that $\{X_i \mid (q, S) \models \varphi_i\} \subseteq q$. Consequently, we have $f(\vec{W}^{t'}) \subseteq \vec{W}^{t'}$, which is equivalent to $f(\vec{W}^\infty) \subseteq \vec{W}^\infty$. ∎

4.4 Capturing asynchronous runs using least fixpoints

This section is dedicated to proving the converse direction of the main result, which will allow us to translate any quasi-acyclic lossless-asynchronous automaton into an equivalent formula of the backward μ-fragment (see Proposition 4.6). Our proof builds on two concepts: the invariance of distributed automata under backward bisimulation (stated in Proposition 4.4) and an ad-hoc relation "$\triangleright$" that captures the possible behaviors of a fixed lossless-asynchronous automaton A (in a specific sense described in Lemma 4.5).

We start with the notion of backward bisimulation, which is defined like the standard notion of bisimulation (see, e.g., [BRV02, Def. 2.16] or [BB07, Def. 5]), except that edges are followed in the backward direction. Formally, a *backward bisimulation* between two s-bit labeled digraphs $G = (V^G, R^G, \lambda^G)$ and $G' = (V^{G'}, R^{G'}, \lambda^{G'})$ is a binary relation $B \subseteq V^G \times V^{G'}$ that fulfills the following conditions for all $vv' \in B$:

a. $\lambda^G(v) = \lambda^{G'}(v')$,

b. if $uv \in R^G$, then there exists $u' \in V^{G'}$ such that $u'v' \in R^{G'}$ and $uu' \in B$, and, conversely,

c. if $u'v' \in R^{G'}$, then there exists $u \in V^G$ such that $uv \in R^G$ and $uu' \in B$.

We say that the pointed digraphs $G[v]$ and $G'[v']$ are *backward bisimilar* if there exists such a backward bisimulation B relating v and v'. It is easy to see that distributed automata cannot distinguish between backward bisimilar structures:

Proposition 4.4 .
▸ Distributed automata are invariant under backward bisimulation. That is, for every automaton A, if two pointed digraphs $G[v]$ and $G'[v']$ are backward bisimilar, then A accepts $G[v]$ if and only if it accepts $G'[v']$. ◂

Proof. Let B be a backward bisimulation between G and G' such that $vv' \in B$. Since acceptance is defined with respect to the synchronous behavior of the automaton, we need only consider the synchronous runs $\rho = (\rho_0, \rho_1, \dots)$ and $\rho' = (\rho'_0, \rho'_1, \dots)$ of A on G and G', respectively. Now, given that the FIFO buffers on the edges of the digraphs merely contain the current state of their incoming node, it is straightforward to prove by induction on t that every pair of nodes $uu' \in B$ satisfies $\rho_t(u) = \rho'_t(u')$ for all $t \in \mathbb{N}$. ∎

We now turn to the mentioned relation "$\triangleright$", which is defined with respect to a fixed automaton. For the remainder of this section, let A denote an automaton (Q, δ_0, δ, F), and let $\mathfrak{Q}$ denote its set of traces. The relation $\triangleright \subseteq (2^{\mathfrak{Q}} \times \mathfrak{Q})$ specifies whether, in a lossless-asynchronous environment, a given trace σ can be traversed by a node whose incoming neighbors traverse the traces of a given set $\mathfrak{S}$. Loosely speaking, the intended meaning of $\mathfrak{S} \triangleright \sigma$ ("$\mathfrak{S}$ enables σ") is the following: Take an appropriately chosen digraph under some lossless-asynchronous timing τ, and observe the corresponding run of A up to a specific time t; if node v was initially in state $\sigma.\mathrm{first}$ and at time t it has *seen* its incoming neighbors traversing precisely the traces in $\mathfrak{S}$, then it is possible for τ to be such that at time t, node v has traversed exactly the trace σ. This relation can be defined inductively: As the base case, we specify that for every $q \in Q$ and $S \subseteq Q$, we have $S \triangleright q.\mathrm{pushlast}(\delta(q, S))$. For the inductive clause, consider a trace $\sigma \in \mathfrak{Q}$ and two finite (possibly equal) sets of traces $\mathfrak{S}, \mathfrak{S}' \subseteq \mathfrak{Q}$ such that the traces in $\mathfrak{S}'$ can be obtained by appending at most one state to the traces in $\mathfrak{S}$. More precisely, if $\pi \in \mathfrak{S}$, then $\pi.\mathrm{pushlast}(p) \in \mathfrak{S}'$ for some $p \in Q$, and conversely, if $\pi' \in \mathfrak{S}'$, then $\pi' = \pi.\mathrm{pushlast}(\pi'.\mathrm{last})$ for some $\pi \in \mathfrak{S}$. We shall denote this auxiliary relation by $\mathfrak{S} \rightrightarrows \mathfrak{S}'$. If it holds, then $\mathfrak{S} \triangleright \sigma$ implies $\mathfrak{S}' \triangleright \sigma.\mathrm{pushlast}(q)$, where $q = \delta(\sigma.\mathrm{last}, \{\pi'.\mathrm{last} \mid \pi' \in \mathfrak{S}'\})$.

The next step is to show (in Lemma 4.5) that our definition of "$\triangleright$" does indeed capture the intuition given above. To formalize this, we first introduce two further pieces of terminology.

First, the notions of configuration and run can be enriched to facilitate discussions about the past. Let $\rho = (\rho_0, \rho_1, \dots)$ be a run of A on a digraph $G = (V^G, R^G, \lambda^G)$ (timed by some timing τ). The corresponding *enriched run* is the sequence $\hat{\rho} = (\hat{\rho}_0, \hat{\rho}_1, \dots)$ of *enriched configurations* that we obtain from ρ by requiring each node to remember the entire trace it has traversed so far. Formally, for $t \in \mathbb{N}$, $v \in V^G$ and $e \in R^G$,

$$\hat{\rho}_0(v) = \rho_0(v), \quad \hat{\rho}_{t+1}(v) = \hat{\rho}_t(v).\mathrm{pushlast}(\rho_{t+1}(v)) \quad \text{and} \quad \hat{\rho}_t(e) = \rho_t(e).$$

Second, we will need to consider finite segments of timings and enriched runs. A *lossless-asynchronous timing segment* of a digraph G is a sequence $\tau = (\tau_1, \dots, \tau_r)$ that could be extended to a whole lossless-asynchronous timing $(\tau_1, \dots, \tau_r, \tau_{r+1}, \dots)$.

Likewise, for an initial enriched configuration $\hat{\rho}_0$ of G, the corresponding *enriched run segment* timed by τ is the sequence $(\hat{\rho}_0, \dots, \hat{\rho}_r)$, where each $\hat{\rho}_{t+1}$ is computed from $\hat{\rho}_t$ and τ_{t+1} in the same way as for an entire enriched run.

Equipped with the necessary terminology, we can now state and prove a (slightly technical) lemma that will allow us to derive benefit from the relation "$\triangleright$". This lemma essentially states that if $\mathfrak{S} \triangleright \sigma$ holds and we are given enough nodes that traverse the traces in $\mathfrak{S}$, then we can take those nodes as the incoming neighbors of a new node v and delay the messages received by v in such a way that v traverses σ, without losing any messages.

Lemma 4.5 .

▶ For every trace $\sigma \in \mathfrak{Q}$ and every finite (possibly empty) set of traces $\mathfrak{S} = \{\pi_1, \dots, \pi_\ell\} \subseteq \mathfrak{Q}$ that satisfy the relation $\mathfrak{S} \triangleright \sigma$, there are lower bounds $m_1, \dots, m_\ell \in \mathbb{N}_+$ such that the following statement holds true:

For any $n_1, \dots, n_\ell \in \mathbb{N}_+$ satisfying $n_i \geqslant m_i$, let G be a digraph consisting of the nodes $(u_i^j)_{i,j}$ and v, and the edges $(u_i^j v)_{i,j}$, with index ranges $1 \leqslant i \leqslant \ell$ and $1 \leqslant j \leqslant n_i$. If we start from the enriched configuration $\hat{\rho}_0$ of G, where

$$\hat{\rho}_0(u_i^j) = \pi_i, \qquad \hat{\rho}_0(u_i^j v) = \pi_i \quad \text{and} \quad \hat{\rho}_0(v) = \sigma.\mathsf{first},$$

then we can construct a (nonempty) lossless-asynchronous timing segment $\tau = (\tau_1, \dots, \tau_r)$ of G, where $\tau_t(u_i^j) = 0$ and $\tau_t(v) = 1$ for $1 \leqslant t \leqslant r$, such that the corresponding enriched run segment $\hat{\rho} = (\hat{\rho}_0, \dots, \hat{\rho}_r)$ timed by τ satisfies

$$\hat{\rho}_{r-1}(u_i^j v) = \pi_i.\mathsf{last} \quad \text{and} \quad \hat{\rho}_r(v) = \sigma. \qquad ◀$$

Proof. We proceed by induction on the definition of "$\triangleright$". In the base case, where $\mathfrak{S} = \{p_1, \dots, p_\ell\} \subseteq Q$ and $\sigma = q.\mathsf{pushlast}(\delta(q, \mathfrak{S}))$ for some $q \in Q$, the statement holds with $m_1 = \dots = m_\ell = 1$. This is witnessed by a timing segment $\tau = (\tau_1)$, where $\tau_1(u_i^j) = 0$, $\tau_1(v) = 1$, and $\tau_1(u_i^j v)$ can be chosen as desired.

For the inductive step, assume that the statement holds for σ and $\mathfrak{S} = \{\pi_1, \dots, \pi_\ell\}$ with some values $m_1, \dots, m_\ell$. Now consider any other set of traces $\mathfrak{S}' = \{\pi'_1, \dots, \pi'_{\ell'}\}$ such that $\mathfrak{S} \rightrightarrows \mathfrak{S}'$, and let $\sigma' = \sigma.\mathsf{pushlast}(q)$, where $q = \delta(\sigma.\mathsf{last}, \{\pi'_k.\mathsf{last} \mid \pi'_k \in \mathfrak{S}'\})$. Since $\mathfrak{S} \triangleright \sigma$, we have $\mathfrak{S}' \triangleright \sigma'$. The remainder of the proof consists in showing that the statement also holds for σ' and $\mathfrak{S}'$ with some large enough integers $m'_1, \dots, m'_{\ell'}$. Let us fix $m'_k = \sum\{m_i \mid \pi_i.\mathsf{pushlast}(\pi'_k.\mathsf{last}) = \pi'_k\}$. (As there is no need to find minimal values, we opt for easy expressibility.)

Given any numbers $n'_1, \dots, n'_{\ell'}$ with $n'_k \geqslant m'_k$, we choose suitable values $n_1, \dots, n_\ell$ with $n_i \geqslant m_i$, and consider the corresponding digraph G described in the lemma. Because we have $\mathfrak{S} \rightrightarrows \mathfrak{S}'$, we can assign to each node u_i^j a state p_i^j such that $\pi_i.\mathsf{pushlast}(p_i^j) \in \mathfrak{S}'$. Moreover, provided our choice of $n_1, \dots, n_\ell$ was adequate, we can also ensure that for each $\pi'_k \in \mathfrak{S}'$, there are exactly n'_k nodes u_i^j such that $\pi_i.\mathsf{pushlast}(p_i^j) = \pi'_k$. (Note that nodes with distinct traces $\pi_i, \pi_{i'} \in \mathfrak{S}$ might be mapped to the same trace $\pi'_k \in \mathfrak{S}'$, in case $\pi_{i'} = \pi_i p_i^j$.) It is straightforward to verify that such a choice of numbers and such an assignment of states are always possible, given the lower bounds $m'_1, \dots, m'_{\ell'}$ specified above.

Let us now consider the lossless-asynchronous timing segment $\tau = (\tau_1, \dots, \tau_r)$ and the corresponding enriched run segment $\hat{\rho} = (\hat{\rho}_0, \dots, \hat{\rho}_r)$ provided by the induction hypothesis. Since the popfirst operation has no effect on a trace of length 1, we may assume without loss of generality that $\tau_t(u_i^j v) = 0$ if $\hat{\rho}_{t-1}(u_i^j v)$ has length 1, for $t < r$.

Consequently, if we start from the alternative enriched configuration $\hat{\rho}'_0$, where

$$\hat{\rho}'_0(u_i^j) = \pi_i.\mathsf{pushlast}(p_i^j), \quad \hat{\rho}'_0(u_i^j v) = \pi_i.\mathsf{pushlast}(p_i^j) \quad \text{and} \quad \hat{\rho}'_0(v) = \sigma.\mathsf{first},$$

then the corresponding enriched run segment $(\hat{\rho}'_0, \dots, \hat{\rho}'_r)$ timed by τ can be derived from $\hat{\rho}$ by simply applying "$\mathsf{pushlast}(p_i^j)$" to $\hat{\rho}_t(u_i^j)$ and $\hat{\rho}_t(u_i^j v)$, for $t < r$. We thus get

$$\hat{\rho}'_{r-1}(u_i^j v) = \pi_i.\mathsf{last}.\mathsf{pushlast}(p_i^j) \quad \text{and} \quad \hat{\rho}'_r(v) = \sigma.$$

We may also assume without loss of generality that $\tau_r(u_i^j v) = 1$ if $\hat{\rho}'_{r-1}(u_i^j v)$ has length 2, since this does not affect $\hat{\rho}$ and lossless-asynchrony is ensured by $\tau_r(v) = 1$. Hence, it suffices to extend τ by an additional map τ_{r+1}, where $\tau_{r+1}(u_i^j) = 0$, $\tau_{r+1}(v) = 1$, and $\tau_{r+1}(u_i^j v)$ can be chosen as desired. The resulting enriched run segment $(\hat{\rho}'_0, \dots, \hat{\rho}'_{r+1})$ satisfies

$$\begin{aligned} &\hat{\rho}'_r(u_i^j v) = p_i^j = \pi'_k.\mathsf{last} \quad (\text{for some } \pi'_k \in \mathfrak{S}') \quad \text{and} \\ &\hat{\rho}'_{r+1}(v) = \sigma.\mathsf{pushlast}(q) = \sigma'. \end{aligned}$$

■

Finally, we can put all the pieces together and prove the converse direction of Theorem 4.2:

Proposition 4.6 ($[\![\text{la-QDA}]\!]_{@\mathrm{DG}_s^1} \subseteq [\![\Sigma_1^{\mu}(\overleftarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}_s^1}$).
▸ For every quasi-acyclic lossless-asynchronous automaton, we can effectively construct an equivalent formula of the backward μ-fragment. ◂

Proof. Assume that $\mathcal{A} = (Q, \delta_0, \delta, F)$ is a quasi-acyclic lossless-asynchronous automaton over s-bit labeled digraphs. Since it is quasi-acyclic, its set of traces $\mathfrak{Q}$ is finite, and thus we can afford to introduce a separate set variable X_σ for each trace $\sigma \in \mathfrak{Q}$. Making use of the relation "$\triangleright$", we convert $\mathcal{A}$ into an equivalent formula $\varphi = \mu\big[X_1, (X_\sigma)_{\sigma\in\mathfrak{Q}}\big].\big[\varphi_1, (\varphi_\sigma)_{\sigma\in\mathfrak{Q}}\big]$ of the backward μ-fragment, where

$$\varphi_1 = \bigvee_{\substack{\sigma\in\mathfrak{Q} \\ \sigma.\mathsf{last}\in F}} X_\sigma, \tag{a}$$

$$\varphi_q = \bigvee_{\substack{x\in 2^s \\ \delta_0(x)=q}} \Big(\bigwedge_{x(i)=1} P_i \wedge \bigwedge_{x(i)=0} \neg P_i \Big) \quad \text{for } q \in Q, \quad \text{and} \tag{b}$$

$$\varphi_\sigma = X_{\sigma.\mathsf{first}} \wedge \bigvee_{\substack{\mathfrak{S}\subseteq\mathfrak{Q} \\ \mathfrak{S}\,\triangleright\,\sigma}} \Big(\big(\bigwedge_{\pi\in\mathfrak{S}} \overleftarrow{\Diamond} X_\pi \big) \wedge \big(\overleftarrow{\Box} \bigvee_{\pi\in\mathfrak{S}} X_\pi \big) \Big) \quad \text{for } \sigma \in \mathfrak{Q} \text{ with } |\sigma| \geq 2. \tag{c}$$

Note that this formula can be constructed effectively because an inductive computation of "$\triangleright$" must terminate after at most $|\mathfrak{Q}| \cdot 2^{|\mathfrak{Q}|}$ iterations.

To prove that φ is indeed equivalent to $\mathcal{A}$, let us consider an arbitrary s-bit labeled digraph $G = (V^G, R^G, \lambda^G)$ and the corresponding least fixpoint $\vec{U} = (U_1, (U_\sigma)_{\sigma\in\mathfrak{Q}}) \in (2^{V^G})^{|\mathfrak{Q}|+1}$ of the operator f associated with $(\varphi_1, (\varphi_\sigma)_{\sigma\in\mathfrak{Q}})$.

The easy direction is to show that for all nodes $v \in V^G$, if $\mathcal{A}$ accepts $G[v]$, then $G[v]$ satisfies φ. For that, it suffices to consider the synchronous enriched run $\hat{\rho} = (\hat{\rho}_0, \hat{\rho}_1, \dots)$ of $\mathcal{A}$ on G. (Any other run timed by a lossless-asynchronous timing would exhibit the same acceptance behavior.) As in the proof of Proposition 4.4, we can simply ignore the FIFO buffers on the edges of G because $\hat{\rho}_t(uv) = \hat{\rho}_t(u).\mathsf{last}$.

Using this, a straightforward induction on t shows that every node $v \in V^G$ satisfies $\{\hat{\rho}_t(u) \mid uv \in R^G\} \triangleright \hat{\rho}_{t+1}(v)$ for all $t \in \mathbb{N}$. (For $t = 0$, the claim follows from the base case of the definition of "$\triangleright$"; for the step from t to $t + 1$, we can immediately apply the inductive clause of the definition.) This in turn allows us to prove that each node v is contained in all the components of $\vec{U}$ that correspond to a trace traversed by v in $\hat{\rho}$, i.e., $v \in U_{\hat{\rho}_t(v)}$ for all $t \in \mathbb{N}$. Naturally, we proceed again by induction: For $t = 0$, we have $\hat{\rho}_0(v) = \delta_0(\lambda^G(v)) \in Q$, hence the subformula $\varphi_{\hat{\rho}_0(v)}$ defined in equation (b) holds at v, and thus $v \in U_{\hat{\rho}_0(v)}$. For the step from t to $t + 1$, we need to distinguish two cases. If $\hat{\rho}_{t+1}(v)$ is of length 1, then it is equal to $\hat{\rho}_t(v)$, and there is nothing new to prove. Otherwise, we must consider the appropriate subformula $\varphi_{\hat{\rho}_{t+1}(v)}$ given by equation (c). We already know from the base case that the conjunct $X_{\hat{\rho}_{t+1}(v).\mathrm{first}} = X_{\hat{\rho}_0(v)}$ holds at v, with respect to any variable assignment that interprets each X_σ as U_σ. Furthermore, by the induction hypothesis, $X_{\hat{\rho}_t(u)}$ holds at every incoming neighbor u of v. Since $\{\hat{\rho}_t(u) \mid uv \in R^G\} \triangleright \hat{\rho}_{t+1}(v)$, we conclude that the second conjunct of $\varphi_{\hat{\rho}_{t+1}(v)}$ must also hold at v, and thus $v \in U_{\hat{\rho}_{t+1}(v)}$. Finally, assuming A accepts $G[v]$, we know by definition that $\hat{\rho}_t(v).\mathrm{last} \in F$ for some $t \in \mathbb{N}$. Since $v \in U_{\hat{\rho}_t(v)}$, this implies that the subformula φ_1 defined in equation (a) holds at v, and therefore that $G[v]$ satisfies φ.

For the converse direction of the equivalence, we have to overcome the difficulty that φ is more permissive than A, in the sense that a node v might lie in U_σ, and yet not be able to follow the trace σ under any timing of G. Intuitively, the reason why we still obtain an equivalence is that A cannot take advantage of all the information provided by any particular run, because it must ensure that for *all* digraphs, its acceptance behavior is independent of the timing. It turns out that even if v cannot traverse σ, some other node v' in an indistinguishable digraph will be able to do so. More precisely, we will show that

> if $v \in U_\sigma$, then there exists a pointed digraph $G'[v']$, backward bisimilar to $G[v]$, and a lossless-asynchronous timing τ' of G', such that $\hat{\rho}'_t(v') = \sigma$ for some $t \in \mathbb{N}$, $(*)$

where $\hat{\rho}'$ is the enriched run of A on G' timed by τ'. Now suppose that $G[v]$ satisfies φ. By equation (a), this means that $v \in U_\sigma$ for some trace σ such that $\sigma.\mathrm{last} \in F$. Consequently, A accepts the pointed digraph $G'[v']$ postulated in $(*)$, based on the claim that v' traverses σ under timing τ' and the fact that A is lossless-asynchronous. Since $G[v]$ and $G'[v']$ are backward bisimilar, it follows from Proposition 4.4 that A also accepts $G[v]$.

It remains to verify $(*)$. We achieve this by computing the least fixpoint $\vec{U}$ inductively and proving the statement by induction on the sequence of approximants $(\vec{U}^0, \vec{U}^1, \dots)$. Note that we do not need to consider the limit case, since $\vec{U} = \vec{U}^n$ for some $n \in \mathbb{N}$.

The base case is trivially true because all the components of $\vec{U}^0$ are empty. Furthermore, if σ consists of a single state q, then we do not even need to argue by induction, as it is evident from equation (b) that for all $j \geqslant 1$, node v lies in U^j_q precisely when $\delta_0(\lambda^G(v)) = q$. It thus suffices to set $G'[v'] = G[v]$ and choose the timing τ' arbitrarily. Clearly, we have $\hat{\rho}'_0(v') = \delta_0(\lambda^G(v)) = q$ if $v \in U^j_q$.

On the other hand, if σ is of length at least 2, we must assume that statement $(*)$ holds for the components of $\vec{U}^j$ in order to prove it for U^{j+1}_σ. To this end, consider an arbitrary node $v \in U^{j+1}_\sigma$. By the first conjunct in (c) and the preceding remarks regarding the trivial cases, we know that $\delta_0(\lambda^G(v)) = \sigma.\mathrm{first}$ (and incidentally that

$j \geqslant 1$). Moreover, the second conjunct ensures the existence of a (possibly empty) set of traces $\mathfrak{S}$ that satisfies $\mathfrak{S} \triangleright \sigma$ and that represents a "projection" of v's incoming neighborhood at stage j. By the latter we mean that for all $\pi \in \mathfrak{S}$, there exists $u \in V^G$ such that $uv \in R^G$ and $u \in U^j_\pi$, and conversely, for all $u \in V^G$ with $uv \in R^G$, there exists $\pi \in \mathfrak{S}$ such that $u \in U^j_\pi$.

Now, for each trace $\pi \in \mathfrak{S}$ and each incoming neighbor u of v that is contained in U^j_π, the induction hypothesis provides us with a pointed digraph $G'_{u:\pi}[u'_\pi]$ and a corresponding timing $\tau'_{u:\pi}$, as described in $(*)$. We make $n_{u:\pi} \in \mathbb{N}$ distinct copies of each such digraph $G'_{u:\pi}$. From this, we construct $G' = (V^{G'}, R^{G'}, \lambda^{G'})$ by taking the disjoint union of all the $\sum n_{u:\pi}$ digraphs, and adding a single new node v' with $\lambda^{G'}(v') = \lambda^G(v)$, together with all the edges of the form $u'_\pi v'$ (i.e., one such edge for each copy of every u'_π). Given that every $G'_{u:\pi}[u'_\pi]$ is backward bisimilar to $G[u]$, we can guarantee that the same holds for $G'[v']$ and $G[v]$ by choosing the numbers of digraph copies in G' such that each incoming neighbor u of v is represented by at least one incoming neighbor of v'. That is, for every u, we require that $n_{u:\pi} \geqslant 1$ for some π.

Finally, we construct a suitable lossless-asynchronous timing τ' of G', which proceeds in two phases to make v' traverse σ in the corresponding enriched run $\hat{\rho}'$. In the first phase, where $0 < t \leqslant t_1$, node v' remains inactive, which means that every τ_t assigns 0 to v' and its incoming edges. The state of v' at time t_1 is thus still $\sigma.\mathrm{first}$. Meanwhile, in every copy of each digraph $G'_{u:\pi}$, the nodes and edges behave according to timing $\tau'_{u:\pi}$ until the respective copy of u'_π has completely traversed π, whereupon the entire subgraph becomes inactive. By choosing t_1 large enough, we make sure that the FIFO buffer on each edge of the form $u'_\pi v'$ contains precisely π at time t_1. In the second phase, which lasts from $t_1 + 1$ to t_2, the only active parts of G' are v' and its incoming edges. Since the number $n_{u:\pi}$ of copies of each digraph $G'_{u:\pi}$ can be chosen as large as required, we stipulate that for every trace $\pi \in \mathfrak{S}$, the sum of $n_{u:\pi}$ over all u exceeds the lower bound m_π that is associated with π when invoking Lemma 4.5 for σ and $\mathfrak{S}$. Applying that lemma, we obtain a lossless-asynchronous timing segment of the subgraph induced by v' and its incoming neighbors. This segment determines our timing τ' between $t_1 + 1$ and t_2 (the other parts of G' being inactive), and gives us $\hat{\rho}'_{t_2}(v') = \sigma$, as desired. Naturally, the remainder of τ', starting at $t_2 + 1$, can be chosen arbitrarily, so long as it satisfies the properties of a lossless-asynchronous timing.

As a closing remark, note that the pointed digraph $G'[v']$ constructed above is very similar to the standard unraveling of $G[v]$ into a (possibly infinite) tree. (The set of nodes of that tree-unraveling is precisely the set of all directed paths in G that start at v; see, e.g., [BRV02, Def. 4.51] or [BB07, § 3.2]). However, there are a few differences: First, we do the unraveling backwards, because we want to generate a backward bisimilar structure, where all the edges point toward the root. Second, we may duplicate the incoming neighbors (i.e., children) of each node in the tree, in order to satisfy the lower bounds imposed by Lemma 4.5. Third, we stop the unraveling process at a finite depth (not necessarily the same for each subtree), and place a copy of the original digraph G at every leaf. ∎

Chapter based on the conference paper [KR17].

5
Emptiness Problems

This chapter is concerned with the decidability of the emptiness problem for several classes of nonlocal distributed automata. Given such an automaton, the task is to decide algorithmically whether it accepts on at least one input digraph. For our main variants of local automata, we can easily determine if this is possible, simply on the basis of their logical characterizations: emptiness is decidable for LDA's because they are effectively equivalent to $\overleftarrow{\text{ML}}$, for which the (finite) satisfiability problem is known to be PSPACE-complete; on the other hand, it is undecidable for ALDA_g's because they are effectively equivalent to MSOL, for which (finite) satisfiability is undecidable. We have also shown in Section 3.5, that the corresponding problem for NLDA_g's is decidable, using a simple finite-model argument. Furthermore, by the results on nonlocal automata presented in Chapter 4, we know that emptiness is decidable for a-QDA's and la-QDA's, since (finite) satisfiability for the (backward) μ-calculus is EXPTIME-complete. However, for nonlocal automata in general, the decidability question has been left open by Kuusisto in [Kuu13a]. Indeed, since the logical characterization given there is in terms of the newly introduced modal substitution calculus (for which no decidability results have been previously established), it does not provide us with an immediate answer. Here, we obtain a negative answer for the general case and also consider the question for three subclasses of nonlocal distributed automata.

Our first variant, dubbed forgetful automata, is characterized by the fact that nodes can see their incoming neighbors' states but cannot remember their own state. Although this restriction might seem very artificial, it bears an intriguing connection to classical automata theory: forgetful distributed automata turn out to be equivalent to finite word automata (and hence MSOL) when restricted to pointed dipaths, but strictly more expressive than finite tree automata (and hence MSOL) when restricted to pointed ordered ditrees. As shown in [Kuu13a, Prp. 8], the situation is different on arbitrary digraphs, where distributed automata (and hence forgetful ones) are unable to recognize non-reachability properties that can be easily expressed in MSOL. Hence, none of the two formalisms can simulate the other in general. However, while satisfiability for MSOL is undecidable, we obtain a LOGSPACE algorithm that decides the emptiness problem for forgetful distributed automata.

The preceding decidability result begs the question of what happens if we drop the forgetfulness condition. Motivated by the equivalence of finite word automata and forgetful distributed automata, we first investigate this question when restricted to dipaths. In sharp contrast to the forgetful case, we find that for arbitrary distributed automata, it is undecidable whether an automaton accepts on some dipath. Although our proof follows the standard approach of simulating a Turing machine, it has an unusual twist: we exchange the roles of space and time, in the sense that the space of the simulated Turing machine M is encoded into the time of the simulating distributed automaton A, and conversely, the time of M is encoded into the space of A. To lift this result to arbitrary digraphs, we introduce the class of monovisioned distributed automata, where nodes enter a rejecting sink state as soon as they see more than one state in their incoming neighborhood. For every distributed automaton A, one can construct a monovisioned automaton A' that satisfies the emptiness property if and only if A does so on dipaths. Hence, the emptiness problem is undecidable for monovisioned automata, and thus also in general.

Our third and last class consists of the quasi-acyclic distributed automata. The motivation for considering this particular class is threefold. First, quasi-acyclicity may be seen as a natural intermediate stage between local and nonlocal distributed automata, because local automata (for which the emptiness problem is decidable) can be characterized as those automata whose state diagram is acyclic as long as we ignore sink states (see Section 2.7). Second, the Turing machine simulation mentioned above makes crucial use of directed cycles in the diagram of the simulating automaton, which suggests that cycles might be the source of undecidability. Third, the notion of quasi-acyclic state diagrams also plays a major role in Chapter 4, where it serves as an ingredient for a-QDA's and la-QDA's (for which the emptiness problem is also decidable). However, contrary to what one might expect from these clues, we show that quasi-acyclicity alone is not sufficient to make the emptiness problem decidable, thereby giving an alternative proof of undecidability for the general case.

The remainder of this chapter is organized as follows: We first introduce some formal definitions in Section 5.1 and establish the connections between forgetful distributed automata and classical word and tree automata in Section 5.2. Then, in Section 5.3, we show the positive decidability result for forgetful automata. Finally, we establish the negative results for monovisioned automata in Section 5.4 and for quasi-acyclic automata in Section 5.5.

5.1 Preliminaries

Given a distributed automaton A, the (general) *emptiness problem* consists in deciding effectively whether the language of A is nonempty, i.e., whether there is a pointed digraph $G[v]$ that is accepted by A. Similarly, the *dipath-emptiness problem* is to decide whether A accepts some pointed dipath.

We now define forgetful distributed automata, which are characterized by the fact that in each communication round, the nodes of the input digraph can see their neighbors' states but cannot remember their own state. As this entails that they are not able to access their own label by storing it in their state, we instead let them reread that label in each round.

Definition 5.1 (Forgetful distributed automaton).
▸ A *forgetful distributed automaton* (FDA) over Σ-labeled, r-relational digraphs is a

tuple $A = (Q, q_0, (\delta_a)_{a \in \Sigma}, F)$, where Q is a finite nonempty set of states, $q_0 \in Q$ is an *initial state*, $\delta_a \colon (2^Q)^r \to Q$ is a *transition function associated with label* $a \in \Sigma$, and $F \subseteq Q$ is a set of accepting states. ◀

The semantics is completely analogous to the one defined in Section 2.7, for the unrestricted automata of Definition 2.3. For a given Σ-labeled, r-relational digraph G, the run ρ of A on G is the infinite sequence of configurations $(\rho_0, \rho_1, \rho_2, \dots)$, which are defined inductively as follows, for $t \in \mathbb{N}$ and $v \in V^G$:

$$\rho_0(v) = q_0 \qquad \text{and} \qquad \rho_{t+1}(v) = \delta_{\lambda^G(v)}\Big(\big(\{\rho_t(u) \mid uv \in R_i^G\}\big)_{1 \leqslant i \leqslant r}\Big).$$

The definition of acceptance remains exactly the same as in Section 2.7, i.e., for $v \in V^G$, the pointed digraph $G[v]$ is accepted by A if and only if there exists $t \in \mathbb{N}$ such that $\rho_t(v) \in F$.

5.2 Comparison with classical automata

The purpose of this section is to motivate our interest in forgetful distributed automata by establishing their connection with classical word and tree automata.

Proposition 5.2 ($[\![\text{FDA}]\!]_{@\text{DIPATH}_\Sigma} = [\![\text{MSOL}]\!]_{@\text{DIPATH}_\Sigma}$).
▶ When restricted to the class of pointed dipaths, forgetful distributed automata are equivalent to finite word automata, and thus to MSOL. ◀

Proof. Let us denote a (deterministic) finite word automaton over some finite alphabet Σ by a tuple $B = (P, p_0, \tau, H)$, where P is the set of states, p_0 is the initial state, $\tau \colon P \times \Sigma \to P$ is the transition function, and H is the set of accepting states.

Given such a word automaton B, we construct a forgetful distributed automaton $A = (Q, q_0, (\delta_a)_{a \in \Sigma}, F)$ that simulates B on Σ-labeled dipaths. For this, it suffices to set $Q = P \cup \{\bot\}$, $q_0 = \bot$, $F = H$, and

$$\delta_a(S) = \begin{cases} \tau(p_0, a) & \text{if } S = \varnothing, \\ \tau(p, a) & \text{if } S = \{p\} \text{ for some } p \in P, \\ \bot & \text{otherwise.} \end{cases}$$

When A is run on a dipath, each node v starts in a waiting phase, represented by $\bot$, and remains idle until its predecessor has computed the state p that B would have reached just before reading the local letter a of v. (If there is no predecessor, p is set to p_0.) Then, v switches to the state $\tau(p, a)$ and stays there forever. Consequently, the distinguished last node of the dipath will end up in the state reached by B at the end of the word, and it accepts if and only if B does.

For the converse direction, we convert a given forgetful distributed automaton $A = (Q, q_0, (\delta_a)_{a \in \Sigma}, F)$ into the word automaton $B = (P, p_0, \tau, H)$ with components $P = 2^Q$, $p_0 = \varnothing$, $H = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$, and

$$\tau(p, a) = \{q_0\} \cup \begin{cases} \{\delta_a(\varnothing)\} & \text{if } p = p_0, \\ \{\delta_a(\{q\}) \mid q \in p\} & \text{otherwise.} \end{cases}$$

On any Σ-labeled dipath G, our construction guarantees that the set of states visited by A at the i-th node is equal to the state that B reaches just after processing the

i-th letter of the word associated with G. We can easily verify this by induction on i: At the first node, which is labeled with a_1, automaton A starts in state q_0 and then remains forever in state $\delta_{a_1}(\varnothing)$. Node number $i+1$ also starts in q_0, and transitions to $\delta_{a_{i+1}}(\{q^i_t\})$ at time $t+1$, where a_{i+1} is the node's own label and q^i_t is the state of its predecessor at time t. In agreement with this behavior, we know by the induction hypothesis and the definition of τ that the state of B after reading a_{i+1} is precisely $\{q_0\} \cup \{\delta_{a_{i+1}}(\{q^i_t\}) \mid t \in \mathbb{N}\}$. As a result, the final state reached by B will be accepting if and only if A visits some accepting state at the last node. ∎

A (deterministic, bottom-up) finite tree automaton over Σ-labeled, r-relational ordered ditrees can be defined as a tuple $B = (P, (\tau_k)_{0 \leqslant k \leqslant r}, H)$, where P is a finite nonempty set of states, $\tau_k \colon P^k \times \Sigma \to P$ is a transition function of arity k, and $H \subseteq P$ is a set of accepting states. Such an automaton assigns a state of P to each node of a given pointed ordered ditree, starting from the leaves and working its way up to the root. If node v is labeled with letter a and its k children have been assigned the states $p_1, \dots, p_k$ (following the numbering order of the k first edge relations), then v is assigned the state $\tau_k(p_1, \dots, p_k, a)$. Note that leaves are covered by the special case $k = 0$. Based on this, the pointed ditree is accepted if and only if the state at the root belongs to H. For a more detailed presentation see, e.g., [Löd12, § 3.3].

Proposition 5.3 ($[\![\text{FDA}]\!]_{@\text{ODITREE}^r_\Sigma} \supsetneq [\![\text{MSOL}]\!]_{@\text{ODITREE}^r_\Sigma}$).
▶ When restricted to the class of pointed ordered ditrees, forgetful distributed automata are strictly more expressive than finite tree automata, and thus than MSOL. ◀

Proof. To convert a tree automaton $B = (P, (\tau_k)_{0 \leqslant k \leqslant r}, H)$ into a forgetful distributed automaton $A = (Q, q_0, (\delta_a)_{a \in \Sigma}, F)$ that is equivalent to B over Σ-labeled, r-relational ordered ditrees, we use a simple generalization of the construction in the proof of Proposition 5.2: $Q = P \cup \{\bot\}$, $q_0 = \bot$, $F = H$, and

$$\delta_a(\vec{S}) = \begin{cases} \tau_k(p_1, \dots, p_k, a) & \text{if } \vec{S} = \big(\{p_1\}, \dots, \{p_k\}, \varnothing, \dots, \varnothing\big) \text{ with } p_1, \dots, p_k \in P, \\ \bot & \text{otherwise.} \end{cases}$$

In contrast, a conversion in the other direction is not always possible, as can be seen from the following example on binary ditrees. Consider the forgetful distributed automaton $A' = (\{\bot, \top, \star\}, \bot, \delta, \{\star\})$, with

$$\delta(S_1, S_2) = \begin{cases} \bot & \text{if } S_1 = S_2 = \{\bot\} \\ \top & \text{if } S_1, S_2 \in \{\varnothing, \{\top\}\} \\ \star & \text{otherwise.} \end{cases}$$

When run on an unlabeled, 2-relational ordered ditree, A' accepts at the root precisely if the ditree is *not* perfectly balanced, i.e., if there exists a node whose left and right subtrees have different heights. To achieve this, each node starts in the waiting state $\bot$, where it remains as long as it has two children and those children are also in $\bot$. If the ditree is perfectly balanced, then all the leaves switch permanently from $\bot$ to $\top$ in the first round, their parents do so in the second round, their parents' parents in the third round, and so forth, until the signal reaches the root. Therefore, the root will transition directly from $\bot$ to $\top$, never visiting state $\star$, and hence the pointed ditree is rejected. On the other hand, if the ditree is not perfectly balanced, then there must be some lowermost internal node v that does not have two subtrees of the same height (in particular, it might have only one child). Since its subtrees are perfectly balanced, they behave as in the preceding case. At some point in time, only one of v's children will be in state $\perp$, at which point v will switch to state $\star$. This triggers an upward-propagating chain reaction, eventually causing the root to also visit $\star$, and thus to accept. Note that $\star$ is just an intermediate state; regardless of whether or not the ditree is perfectly balanced, every node will ultimately end up in $\top$.

To prove that A' is not equivalent to any tree automaton, one can simply invoke the pumping lemma for regular tree languages to show that the complement language of A' is not recognizable by any tree automaton. The claim then follows from the fact that regular tree languages are closed under complementation. ∎

5.3 Exploiting forgetfulness

We now give an algorithm deciding the emptiness problem for forgetful distributed automata (on arbitrary digraphs). Its space complexity is linear in the number of states of the given automaton. However, as an uncompressed binary encoding of a distributed automaton requires space exponential in the number of states, this results in LOGSPACE complexity. Obviously, the statement might not hold anymore if the automaton were instead represented by a more compact device, such as a logical formula.

Theorem 5.4 .
▶ We can decide the emptiness problem for forgetful distributed automata with LOGSPACE complexity. ◀

Proof. Let $A = (Q, q_0, (\delta_a)_{a \in \Sigma}, F)$ be some forgetful distributed automaton over Σ-labeled, r-relational digraphs. Consider the infinite sequence of sets of states $S_0, S_1, S_2 \cdots$ such that S_t contains precisely those states that can be visited by A at some node in some digraph at time t. That is, $q \in S_t$ if and only if there exists a pointed digraph $G[v]$ such that $\rho_t(v) = q$, where ρ is the run of A on G. From this point of view, the pointed-digraph language of A is nonempty precisely if there is some $t \in \mathbb{N}$ for which $S_t \cap F \neq \varnothing$.

By definition, we have $S_0 = \{q_0\}$. Furthermore, exploiting the fact that A is forgetful, we can specify a simple function $\Delta\colon 2^Q \to 2^Q$ such that $S_{t+1} = \Delta(S_t)$:

$$\Delta(S) = \left\{ \delta_a(\vec{T}) \;\middle|\; a \in \Sigma \text{ and } \vec{T} \in (2^S)^r \right\}$$

Obviously, $S_{t+1} \subseteq \Delta(S_t)$. To see that $S_{t+1} \supseteq \Delta(S_t)$, assume we are given a pointed digraph $G_q[v_q]$ for each state $q \in S_t$ such that v_q visits q at time t in the run of A on G_q. (Such a pointed digraph must exist by the definition of S_t.) Now, for any $a \in \Sigma$ and $\vec{T} = (T_1, \dots, T_r) \in (2^{S_t})^r$, we construct a new digraph G as follows: Starting with a single a-labeled node v, we add a (disjoint) copy of G_q for each state q that occurs in some set T_k. Then, we add a k-edge from v_q to v if and only if $q \in T_k$. Each node v_q behaves the same way in G as in G_q because v has no influence on its incoming neighbors. Since A is forgetful, the state of v at time $t+1$ depends solely on its own label and its incoming neighbors' states at time t. Consequently, v visits the state $\delta_a(\vec{T})$ at time $t+1$, and thus $\delta_a(\vec{T}) \in S_{t+1}$.

Now, we know that the sequence $S_0, S_1, S_2 \cdots$ must be eventually periodic because its generator function Δ maps the finite set 2^Q to itself. Hence, it suffices to consider the prefix of length $|2^Q|$ in order to determine whether $S_t \cap F \neq \varnothing$ for some $t \in \mathbb{N}$. This leads to the following simple algorithm, which decides the emptiness problem for forgetful automata.

$$
\begin{array}{ll}
\textsc{empty}(A): & S \leftarrow \{q_0\} \\
& \textbf{repeat} \text{ at most } |2^Q| \text{ times}: \\
& \quad S \leftarrow \Delta(S) \\
& \quad \textbf{if } S \cap F \neq \varnothing: \ \textbf{return} \text{ true} \\
& \textbf{return} \text{ false}
\end{array}
$$

It remains to analyze the space complexity of this algorithm. For that, we assume that the binary encoding of A given to the algorithm contains a lookup table for each transition function δ_a and a bit array representing F, which amounts to an asymptotic size of $\Theta\big(|\Sigma| \cdot |2^Q|^r \cdot \log|Q|\big)$ input bits. To implement the procedure $\textsc{empty}$, we need $|Q|$ bits of working memory to represent the set S and another $|Q|$ bits for the loop counter. Furthermore, we can compute $\Delta(S)$ for any given set $S \subseteq Q$ by simply iterating over all $a \in \Sigma$ and $\vec{T} \in (2^Q)^r$, and adding $\delta_a(\vec{T})$ to the returned set if all components of $\vec{T}$ are subsets of S. This requires $\log|\Sigma| + |Q| \cdot r$ additional bits to keep track of the iteration progress, $\Theta\big(\log|\Sigma| + |Q| \cdot r + \log\log|Q|\big)$ bits to store pointers into the lookup tables, and $|Q|$ bits to store the intermediate result. In total, the algorithm uses $\Theta\big(\log|\Sigma| + |Q| \cdot r\big)$ bits of working memory, which is logarithmic in the size of the input. ∎

5.4 Exchanging space and time

In this section, we first show the undecidability of the dipath-emptiness problem for arbitrary distributed automata, and then lift that result to the general emptiness problem.

Theorem 5.5 .
▶ The dipath-emptiness problem for distributed automata is undecidable. ◀

Proof sketch. We proceed by reduction from the halting problem for Turing machines. For our purposes, a Turing machine operates deterministically with one head on a single tape, which is one-way infinite to the right and initially empty. The problem consists of determining whether the machine will eventually reach a designated halting state. We show a way of encoding the computation of a Turing machine M into the run of a distributed automaton A over unlabeled digraphs, such that the language of A contains a pointed dipath if and only if M reaches its halting state.

Note that since dipaths are oriented, the communication between their nodes is only one-way. Hence, we cannot simply represent (a section of) the Turing tape as a dipath. Instead, the key idea of our simulation is to exchange the roles of space and time, in the sense that the space of M is encoded into the time of A, and the time of M into the space of A. Assuming the language of A contains a dipath, we will think of that dipath as representing the timeline of M, such that each node corresponds to a single point in time in the computation of M. Roughly speaking, when running A,

It turns out that this corresponds to a well-known construction in cellular automata theory; see Section 7.2.2.

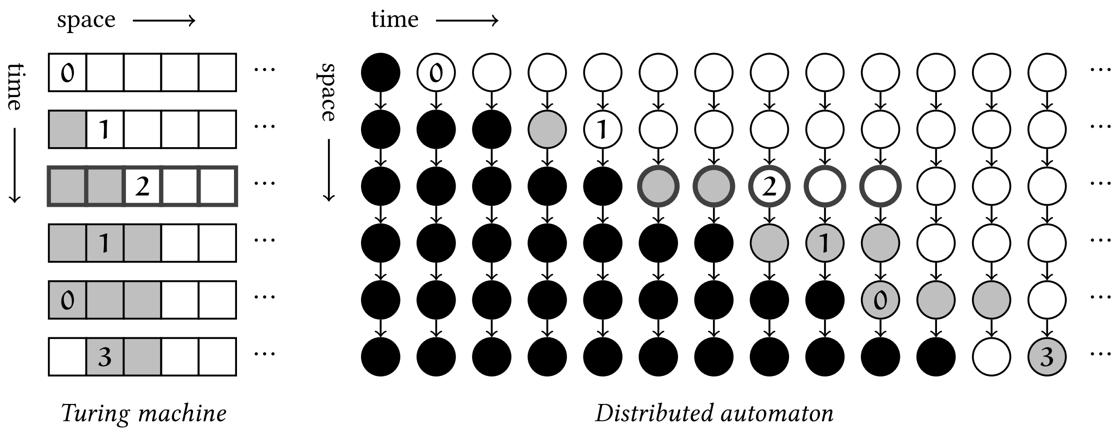

Figure 5.1. Exchanging space and time to prove Theorem 5.5. The left-hand side depicts the computation of a Turing machine with state set $\{0, 1, 2, 3\}$ and tape alphabet $\{\square, \blacksquare\}$. On the right-hand side, this machine is simulated by a distributed automaton run on a dipath. Waiting nodes are represented in black, whereas active nodes display the content of the "currently visited" cell of the Turing machine (i.e., only the third component of the states is shown).

the node v_t corresponding to time t will "traverse" the configuration C_t of M at time t. Here, "traversing" means that the sequence of states of A visited by v_t is an encoding of C_t read from left to right, supplemented with some additional bookkeeping information.

The first element of the dipath, node v_0, starts by visiting a state of A representing an empty cell that is currently read by M in its initial state. Then it transitions to another state that simply represents an empty cell, and remains in such a state forever after. Thus v_0 does indeed "traverse" C_0. We will show that it is also possible for any other node v_t to "traverse" its corresponding configuration C_t, based on the information it receives from v_{t-1}. In order for this to work, we shall give v_{t-1} a head start of two cells, so that v_t can compute the content of cell i in C_t based on the contents of cells $i-1$, i and $i+1$ in C_{t-1}.

Node v_t enters an accepting state of A precisely if it "sees" the halting state of M during its "traversal" of C_t. Hence, A accepts the pointed dipath of length t if and only if M reaches its halting state at time t.

We now describe the inner workings of A in a semi-formal way. In parallel, the reader might want to have a look at Figure 5.1, which illustrates the construction by means of an example. Let M be represented by the tuple $(P, \Gamma, p_0, \square, \tau, p_h)$, where P is the set of states, Γ is the tape alphabet, p_0 is the initial state, $\square$ is the blank symbol, $\tau\colon (P \setminus \{p_h\}) \times \Gamma \to P \times \Gamma \times \{L, R\}$ is the transition function, and p_h is the halting state. From this, we construct A as (Q, q_0, δ, F), with the state set $Q = (\{\bot\} \cup (P \times \Gamma) \cup \Gamma)^3$, the initial state $q_0 = (\bot, \bot, \bot)$, the transition function δ specified informally below, and the accepting set F that contains precisely those states that have p_h in their third component. In keeping with the intuition that each node of the dipath "traverses" a configuration of M, the third component of its state indicates the content of the "currently visited" cell i. The two preceding components keep track of the recent history, i.e., the second component always holds the content of the previous cell $i-1$, and the first component that of $i-2$. In the following explanation, we concentrate on updating the third component, tacitly assuming that the other two are kept up to date. The special symbol $\perp$ indicates that no cell has been "visited", and we say that a node is in the waiting phase while its third component is $\perp$.

In the first round, v_0 sees that it does not have any incoming neighbor, and thus exits the waiting phase by setting its third component to $(p_0, \square)$, and after that, it sets it to $\square$ for the remainder of the run. Every other node v_t remains in the waiting phase as long as its incoming neighbor's second component is $\perp$. This ensures a delay of two cells with respect to v_{t-1}. Once v_t becomes active, given the *current* state (c_1, c_2, c_3) of v_{t-1}, it computes the third component d_3 of its own *next* state (d_1, d_2, d_3) as follows: If none of the components c_1, c_2, c_3 "contain the head of M", i.e., if none of them lie in $P \times \Gamma$, then it simply sets d_3 to be equal to c_2. Otherwise, a computation step of M is simulated in the natural way. For instance, if c_3 is of the form (p, γ), and $\tau(p, \gamma) = (p', \gamma', L)$, then d_3 is set to (p', c_2). This corresponds to the case where, at time $t - 1$, the head of M is located to the right of v_t's next "position" and moves to the left. As another example, if c_2 is of the form (p, γ), and $\tau(p, \gamma) = (p', \gamma', R)$, then d_3 is set to γ'. The remaining cases are handled analogously.

Note that, thanks to the two-cell delay between adjacent nodes, the head of M always "moves forward" in the time of A, although it may move in both directions with respect to the space of M (see Figure 5.1). ■

To infer from Theorem 5.5 that the general emptiness problem for distributed automata is also undecidable, we now introduce the notion of *monovisioned automata*, which have the property that nodes "expect" to see no more than one state in their incoming neighborhood at any given time. More precisely, a distributed automaton $A = (Q, \delta_0, \delta, F)$ is monovisioned if it has a rejecting sink state $q_{\text{rej}} \in Q \setminus F$, such that $\delta(q, S) = q_{\text{rej}}$ whenever $|S| > 1$ or $q_{\text{rej}} \in S$ or $q = q_{\text{rej}}$, for all $q \in Q$ and $S \subseteq Q$. Obviously, for every distributed automaton, we can construct a monovisioned automaton that has the same acceptance behavior on dipaths. Furthermore, as shown by means of the next two lemmas, the emptiness problem for monovisioned automata is equivalent to its restriction to dipaths. All put together, we get the desired reduction from the dipath-emptiness problem to the general emptiness problem.

Lemma 5.6 .

▶ The language of a distributed automaton is nonempty if and only if it contains a pointed ditree. ◀

Proof sketch. We slightly adapt the notion of *tree-unraveling*, which is a standard tool in modal logic (see, e.g., [BRV02, Def. 4.51] or [BB07, § 3.2]). Consider any distributed automaton A. Assume that A accepts some pointed digraph $G[v]$, and let $t \in \mathbb{N}$ be the first point in time at which v visits an accepting state. Based on that, we can easily construct a pointed ditree $G'[v']$ that is also accepted by A. First of all, the root v' of G' is chosen to be a copy of v. On the next level of the ditree, the incoming neighbors of v' are chosen to be fresh copies $u'_1, \dots, u'_n$ of v's incoming neighbors $u_1, \dots, u_n$. Similarly, the incoming neighbors of $u'_1, \dots, u'_n$ are fresh copies of the incoming neighbors of $u_1, \dots, u_n$. If u_i and u_j have incoming neighbors in common, we create distinct copies of those neighbors for u'_i and u'_j. This process is iterated until we obtain a ditree of height t. It is easy to check that v and v' visit the same sequence of states $q_0, q_1, \dots, q_t$ during the first t communication rounds. ■

Lemma 5.7 .
▶ The language of a monovisioned distributed automaton is nonempty if and only if it contains a pointed dipath. ◀

Proof sketch. Consider any monovisioned distributed automaton A whose language is nonempty. By Lemma 5.6, A accepts some pointed ditree $G[v]$. Let $t \in \mathbb{N}$ be the first point in time at which v visits an accepting state. Now, it is easy to prove by induction that for all $i \in \{0, \dots, t\}$, sibling nodes at depth i traverse the same sequence of states $q_0, q_1, \dots, q_{t-i}$ between times 0 and $t - i$, and this sequence does not contain the rejecting state q_{rej}. Thus, A also accepts any dipath from some node at depth t to the root. ∎

5.5 Timing a firework show

We now show that the emptiness problem is undecidable even for quasi-acyclic automata. This also provides an alternative, but more involved undecidability proof for the general case. Notice that our proof of Theorem 5.5 does not go through if we consider only quasi-acyclic automata.

It is straightforward to see that quasi-acyclicity is preserved under a standard product construction, similar to the one employed for finite automata on words. Hence, we have the following closure property, which will be used in the subsequent undecidability proof.

Lemma 5.8 .
▶ The class of languages recognizable by quasi-acyclic distributed automata is closed under union and intersection. ◀

Theorem 5.9 .
▶ The emptiness problem for quasi-acyclic distributed automata is undecidable. ◀

Proof sketch. We show this by reduction from Post's correspondence problem (PCP). An instance P of PCP consists of a collection of pairs of nonempty finite words $(x_i, y_i)_{i \in I}$ over the alphabet $\{0, 1\}$, indexed by some finite set of integers I. It is convenient to view each pair (x_i, y_i) as a domino tile labeled with x_i on the upper half and y_i on the lower half. The problem is to decide if there exists a nonempty sequence $S = (i_1, \dots, i_n)$ of indices in I, such that the concatenations $x_S = x_{i_1} \cdots x_{i_n}$ and $y_S = y_{i_1} \cdots y_{i_n}$ are equal. We construct a quasi-acyclic automaton A whose language is nonempty if and only if P has such a solution S.

Metaphorically speaking, our construction can be thought of as a perfectly timed "firework show", whose only "spectator" will see a putative solution $S = (i_1, \dots, i_n)$, and be able to check whether it is indeed a valid solution of P. Our "spectator" is the distinguished node v_ε of the pointed digraph on which A is run. We assume that v_ε has n incoming neighbors, one for each element of S. Let v_k denote the neighbor corresponding to i_k, for $1 \leqslant k \leqslant n$. Similarly to our proof of Theorem 5.5, we use the time of A to represent the spatial dimension of the words x_S and y_S. On an intuitive level, v_ε will "witness" simultaneous left-to-right traversals of x_S and y_S, advancing by one bit per time step, and it will check that the two words match. It is the task of each node v_k to send to v_ε the required bits of the subwords x_{i_k} and y_{i_k} at the appropriate times. In keeping with the metaphor of fireworks, the correct timing can be achieved by attaching to v_k a carefully chosen "fuse", which is "lit" at time 0. Two separate "fire" signals will travel at different speeds along this (admittedly sophisticated) "fuse", and once they reach v_k, they trigger the "firing" of x_{i_k} and y_{i_k}, respectively.

We now go into more details. Using the labeling of the input graph, the automaton A distinguishes between $2|I| + 1$ different types of nodes: two types i and i' for each index $i \in I$, and one additional type ϵ to identify the "spectator". Motivated by Lemma 5.6, we suppose that the input graph is a pointed ditree, with a very specific shape that encodes a putative solution $S = (i_1, \ldots, i_n)$. An example illustrating the following description of such a ditree-encoding is given in Figure 5.2. Although A is not able to enforce all aspects of this particular shape, we will make sure that it accepts such a structure if its language is nonempty. The root (and distinguished node) v_ϵ is the only node of type ϵ. Its children $v_1, \ldots, v_n$ are of types $i_1, \ldots, i_n$, respectively. The "fuse" attached to each child v_k is a chain of $k - 1$ nodes that represents the multiset of indices occurring in the $(k-1)$-prefix of S. More precisely, there is an induced dipath $v_{k,1} \to \cdots v_{k,k-1} \to v_k$, such that the multiset of types of the nodes $v_{k,1}, \ldots, v_{k,k-1}$ is equal to the multiset of indices occurring in $(i_1, \ldots, i_{k-1})$. We do not impose any particular order on those nodes. Finally, each node of type $i \in I$ also has an incoming chain of nodes of type i' (depicted in gray in Figure 5.2), whose length corresponds exactly to the product of the types occurring on the part of the "fuse" below that node. That is, if we define the alias $v_{k,k} \coloneqq v_k$, then for every node $v_{k,j}$ of type $i \in I$, there is an induced dipath $v_{k,j,1} \to \cdots v_{k,j,\ell} \to v_{k,j}$, where all the nodes $v_{k,j,1}, \ldots, v_{k,j,\ell}$ are of type i', and the number ℓ is equal to the product of the types of the nodes $v_{k,1}, \ldots, v_{k,j-1}$ (which is 1 if $j = 1$). We shall refer to such a chain $v_{k,j,1}, \ldots, v_{k,j,\ell}$ as a "side fuse".

The automaton A has to perform two tasks simultaneously: First, assuming it is run on a ditree-encoding of a sequence S, exactly as specified above, it must verify that S is a valid solution, i.e., that the words x_S and y_S match. Second, it must ensure that the input graph is indeed sufficiently similar to such a ditree-encoding. In particular, it has to check that the "fuses" used for the first task are consistent with each other. Since, by Lemma 5.8, quasi-acyclic distributed automata are closed under intersection, we can consider the two tasks separately, and implement them using two independent automata A_1 and A_2. In the following, we describe both devices in a rather informal manner. The important aspect to note is that they can be easily formalized using quasi-acyclic state diagrams.

We start with A_1, which verifies the solution S. It takes into account only nodes with types in $I \cup \{\epsilon\}$ (thus ignoring the gray nodes in Figure 5.2). At nodes of type $i \in I$, the states of A_1 have two components, associated with the upper and lower halves of the domino (x_i, y_i). If a node of type i sees that it does not have any incoming neighbor, then the upper and lower components of its state immediately start traversing sequences of substates representing the bits of x_i and y_i, respectively. Since those substates must keep track of the respective positions within x_i and y_i, none of them can be visited twice. After that, both components loop forever on a special substate $\top$, which indicates the end of transmission. The other nodes of type i keep each of their two components in a waiting status, indicated by another substate $\bot$, until the corresponding component of their incoming neighbor reaches its last substate before $\top$. This constitutes the aforementioned "fire" signal. Thereupon, they start traversing the same sequences of substates as in the previous case. Note that both components are updated independently of each other, hence there can be an arbitrary time lag between the "traversals" of x_i and y_i. Now, assuming the "fuse"

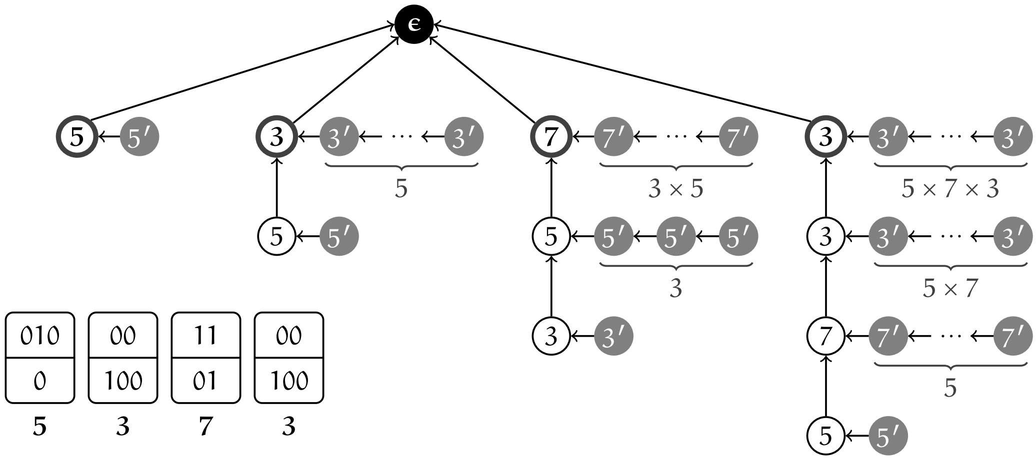

Figure 5.2. Timing a "firework show" to prove Theorem 5.9. The domino tiles on the bottom-left visualize the solution $(5, 3, 7, 3)$ for the instance $\{3 \mapsto (00, 100),\, 5 \mapsto (010, 0),\, 7 \mapsto (11, 01)\}$ of PCP. This solution is encoded into the labeled ditree above, with node types $\epsilon, 3, 5, 7, 3', 5', 7'$. Each domino is represented by a bold-highlighted white node of the appropriate type. The "fuse" of such a bold node consists of the chain of white nodes below it, which lists the indices of the preceding dominos in an arbitrary order. Each white node also has a gray "side fuse" whose length is equal to the product of the white types occurring below that node. The "firework show" observed at the root will feature two simultaneous bitstreams, which both represent the sequence 010001100.

of each node v_k really encodes the multiset of indices occurring in $(i_1, \ldots, i_{k-1})$, the delay accumulated along that "fuse" will be such that v_k starts "traversing" x_{i_k} and y_{i_k} at the points in time corresponding to their respective starting positions within x_S and y_S. That is, for x_{i_k} it starts at time $|x_{i_1} \cdots x_{i_{k-1}}| + 1$, and for y_{i_k} at time $|y_{i_1} \cdots y_{i_{k-1}}| + 1$. Consequently, in each round $t \leqslant \min\{|x_S|, |y_S|\}$, the root v_ϵ receives the t-th bits of x_S and y_S. At most two distinct children send bits at the same time, while the others remain in some state $q \in \{\bot, \top\}^2$. With this, the behavior of A_1 at v_ϵ is straightforward: It enters its only accepting state precisely if all of its children have reached the state $(\top, \top)$ and it has never seen any mismatch between the upper and lower bits.

We now turn to A_2, whose job is to verify that the "fuses" used by A_1 are reliable. Just like A_1, it works under the assumption that the input digraph is a ditree as specified previously, but with significantly reduced guarantees: The root could now have an arbitrary number of children, the "fuses" and "side fuses" could be of arbitrary lengths, and each "fuse" could represent an arbitrary multiset of indices in I. Again using an approach reminiscent of fireworks, we devise a protocol in which each child v will send two distinct signals to the root v_ϵ. The first signal $\uparrow_1$ indicates that the current time t is equal to the product of the types of all the nodes on v's "fuse". Similarly, the second signal $\uparrow_2$ indicates that the current time is equal to that same product multiplied by v's own type. To achieve this, we make use of the "side fuses", along which two additional signals $\leftarrow_1$ and $\leftarrow_2$ are propagated. For each node of type $i \in I$, the nodes of type i' on the corresponding "side fuse" operate in a way such that $\leftarrow_1$ advances by one node per time step, whereas $\leftarrow_2$ is delayed by i time units at every node. Hence, $\leftarrow_1$ travels i times faster than $\leftarrow_2$. Building on that, each node v of type i (not necessarily a child of the root) sends $\uparrow_1$ to its parent, either at time 1, if it does not have any predecessor on the "fuse", or one time unit before receiving $\uparrow_2$ from its predecessor. The latter is possible, because the predecessor also sends a pre-signal $\uparrow_2^{\mathrm{pre}}$ before sending $\uparrow_2$. Then, v checks that signal $\leftarrow_1$ from its "side fuse" arrives exactly at the same time as $\uparrow_2$ from its predecessor, or at time 1 if there is no predecessor. Otherwise, it immediately enters a rejecting state. This will guarantee, by induction, that the length of the "side fuse" is equal to the product of the types on the "fuse" below. Finally, two rounds prior to receiving $\leftarrow_2$, while that signal is still being delayed by the last node on the "side fuse", v first sends the pre-signal $\uparrow_2^{\mathrm{pre}}$, and then the signal $\uparrow_2$ in the following round. For this to work, we assume that each node on the "side fuse" waits for at least two rounds between receiving $\leftarrow_2$ from its predecessor and forwarding the signal to its successor, i.e., all indices in I must be strictly greater than 2. Due to the delay accumulated by $\leftarrow_2$ along the "side fuse", the time at which $\uparrow_2$ is sent corresponds precisely to the length of the "side fuse" multiplied by i.

Without loss of generality, we require that the set of indices I contains only prime numbers (as in Figure 5.2). Hence, by the unique-prime-factorization theorem, each multiset of numbers in I is uniquely determined by the product of its elements. This leads to a simple verification procedure performed by A_2 at the root: At time 1, node v_ϵ checks that it receives $\uparrow_1$ and not $\uparrow_2$. After that, it expects to never again see $\uparrow_1$ without $\uparrow_2$, and remains in a loop as long as it gets either no signal at all or both $\uparrow_1$ and $\uparrow_2$. Upon receiving $\uparrow_2$ alone, it exits the loop and verifies that all of its children have sent both signals, which is apparent from the state of each child. The root rejects immediately if any of the expectations above are violated, or if two nodes with different types send the same signal at the same time. Otherwise, it enters an accepting state after leaving the loop. Now, consider the sequence $T = (t_1, \dots, t_{n+1})$ of rounds in which v_ϵ receives at least one of the signals $\uparrow_1$ and $\uparrow_2$. It is easy to see by induction on T that successful completion of the procedure above ensures that there is a sequence $S = (i_1, \dots, i_n)$ of indices in I with the following properties: For each $k \in \{1, \dots, n\}$, the root has at least one child v_k of type i_k that sends $\uparrow_1$ at time t_k and $\uparrow_2$ at time t_{k+1}, and the "fuse" of v_k encodes precisely the multiset of indices occurring in $(i_1, \dots, i_{k-1})$. Conversely, each child of v_ϵ can be associated in the same manner with a unique element of S.

To conclude our proof, we have to argue that the automaton A, which simulates A_1 and A_2 in parallel, accepts some labeled pointed digraph if and only if P has a solution S. The "if" part is immediate, since, by construction, A accepting a ditree-encoding of S is equivalent to S being a valid solution of P. To show the "only if" part, we start with a pointed digraph accepted by A, and incrementally transform it into a ditree-encoding of a solution S, while maintaining acceptance by A: First of all, by Lemma 5.6, we may suppose that the digraph is a ditree. Its root must be of type ϵ, since A would not accept otherwise. Next, we require that A raises an alarm at nodes that see an unexpected set of states in their incoming neighborhood, and that this alarm is propagated up to the root, which then reacts by entering a rejecting sink state. This ensures that the repartition of types is consistent with our specification; for example, that the children of a node of type i' must be of type i' themselves. We now prune the ditree in such a way that nodes of type i keep at most two children and nodes of type i' keep at most one child. (The behavior of the deleted children must be indistinguishable from the behavior of the remaining children, since otherwise an alarm would be raised.) This leaves us with a ditree corresponding exactly to the input "expected" by the automaton A_2. Since it is accepted by A_2, this ditree must be very close to an encoding of a solution $S = (i_1, \ldots, i_n)$, with the only difference that each element i_k of S may be represented by several nodes $v_k^1, \ldots, v_k^m$. However, we know by construction that A behaves the same on all of these representatives. We can therefore remove the subtrees rooted at $v_k^2, \ldots, v_k^m$, and thus we obtain a ditree-encoding of S that is accepted by A. ■

Chapter based on the preprint [Rei16].

6 Alternation Hierarchies

In this chapter, we transfer the set quantifiers of MSOL to the setting of modal logic and investigate the resulting alternation hierarchies. More precisely, we establish separation results for the hierarchies that one obtains by alternating existential and universal set quantifiers in several logics of the form $\text{MSO}(\Phi)$, where Φ is some variant of modal logic.

Within the context of this thesis, the motivation for such hybrids between modal logic and classical logic stems from their close connection to local distributed automata. By [HJK⁺12, HJK⁺15], LDA's are equivalent to $\overleftarrow{\text{ML}}$ (Theorem 2.5), and by Chapter 3, ALDA_g's are equivalent to MSOL (Theorem 3.13). As mentioned in Section 3.6, the combination of those two results suggests an alternative logical characterization of ALDA_g's using $\text{MSO}(\overleftarrow{\text{ML}}_\text{g})$ instead of MSOL. The equivalence of $\text{MSO}(\overleftarrow{\text{ML}}_\text{g})$ and MSOL can be easily proven by a standard technique that simulates node quantifiers through set quantifiers (see, e.g., [Kuu08, Kuu15, § 3]). Yet in some sense, $\text{MSO}(\overleftarrow{\text{ML}}_\text{g})$ provides a more faithful representation of ALDA_g's because it preserves the expressive power of each quantifier alternation level. For instance, the existential fragment $\Sigma_1^{\text{MSO}}(\overleftarrow{\text{ML}}_\text{g})$ specifies exactly the same digraph languages as NLDA_g's, whereas EMSOL is strictly more powerful (see Section 3.6). Therefore, if we want to precisely examine the power of alternation between nondeterministic decisions and universal branchings in ALDA_g's, then we can do so from a purely logical perspective using $\text{MSO}(\overleftarrow{\text{ML}}_\text{g})$. This has the advantage that, compared to state diagrams, formulas take up less space and are usually easier to manipulate.

As it turns out, the above considerations are closely related to an old problem in modal logic. Already in 1983, van Benthem asked in [Ben83] whether the syntactic hierarchy obtained by alternating existential and universal set quantifiers in $\text{MSO}(\overrightarrow{\text{ML}})$ induces a corresponding hierarchy on the semantic side. Remaining unanswered, the question was raised again by ten Cate in [Cat06], and finally a positive answer was provided by Kuusisto in [Kuu08, Kuu15]: he showed that $\text{MSO}(\overrightarrow{\text{ML}})$ induces an *infinite* hierarchy over pointed digraphs. This tells us that the hierarchy does not completely collapse at some level, but a priori leaves open whether or not each number of quantifier alternations corresponds to a separate semantic level.

In [Cat06] *and* [Kuu08, Kuu15], $\text{MSO}(\overrightarrow{\text{ML}})$ *is called* SOPML *(second-order propositional modal logic).*

Kuusisto's proof builds upon the work of Matz, Schweikardt and Thomas in

[MST02] (elaborating on their previous results in [MT97] and [Sch97]), where they have shown that in the case of MSOL on digraphs, the alternation hierarchy is *strict*. Thus, each additional alternation between the two types of set quantifiers properly extends the family of definable digraph languages. Significantly, this separation also holds on grids, a more restrictive class of structures, where it can be established using techniques from classical automata theory. Furthermore, taken in conjunction with the equivalence of $\mathrm{MSO}(\overleftarrow{\mathrm{ML}}_{\mathrm{g}})$ and MSOL, the result on digraphs immediately implies that the corresponding hierarchy of $\mathrm{MSO}(\overleftarrow{\mathrm{ML}}_{\mathrm{g}})$ is *infinite*. But since the alternation levels of that logic are not the same as those of MSOL, it does not seem obvious how strictness could be inferred.

The present chapter provides an alternative way of transferring the results of Matz, Schweikardt and Thomas to the modal setting. In particular, our method allows to show directly that the set quantifier alternation hierarchies of $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$ and $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ are *strict* over (pointed) digraphs. At first sight, this seems to expand the existing body of knowledge, especially since the strictness question for $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$ has been mentioned as an open problem in [Kuu08] and [Kuu13b]. However, it turns out that in both cases, strictness is actually a consequence of infiniteness [A. Kuusisto, personal communication, 3 March 2016]. Although this observation has so far not been formally published, it appears to be folklore in the model-theory community. Hence, this chapter contributes new proofs to essentially known results. Just as Kuusisto has done in [Kuu08, Kuu15], we use as a starting point the strictness result of [MST02] for MSOL on grids. But from there on, the two proof methods diverge considerably.

To avoid the backward modalities of $\mathrm{MSO}(\overleftarrow{\mathrm{ML}}_{\mathrm{g}})$, we work instead with $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$, which is called SOPMLE *in* [Kuu08, Kuu15]. *By duality, separating one alternation hierarchy also separates the other.*

The original approach of Kuusisto is mainly based on the fact that one can simulate first-order quantifiers by means of set quantifiers, combined with a formula stating that a set is a singleton. As already mentioned, this can be used to show that $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ is equivalent to MSOL. The spirit of the proof in [Kuu08, Kuu15] is essentially the same for $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$, although the details are much more technical, since this logic is less expressive than MSOL on arbitrary pointed structures. It is precisely the use of additional second-order quantifiers that leads to the temporary loss of the specific separation results provided by [MST02].

In contrast, one simple insight will allow us to directly transfer those results: When restricted to the class of grids, $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ and MSOL are more than just equivalent – they are *levelwise* equivalent, and consequently all the separation results shown for MSOL also hold for $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ on grids. This approach is based on the observation that the existential fragment of $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ can simulate another model, called *tiling systems*, which has been shown to be equivalent to the existential fragment of MSOL in [GRST96]. On the basis of this new finding, we can then transfer the given separation results from $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ on grids to other classes of digraphs and other extensions of modal logic, such as $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$. While this works along the same general principle as the *strong first-order reductions* used in [MST02], the additional limitations imposed by modal logic force us to introduce custom encoding techniques that cope with the lack of expressive power.

The remainder of this chapter is organized in a top-down manner. After introducing the necessary notation in Section 6.1, we present the main results in Section 6.2, and almost immediately get to the central proof in Section 6.3. The latter relies on several other propositions, but since those are treated as “black boxes”, the main line of reasoning might be comprehensible without reading any further. We then provide all the missing details in the last two sections, which are independent of each other. Section 6.4 establishes the levelwise equivalence of three different alternation hierarchies on grids, and may thus be interesting on its own. On the other hand, Section 6.5 is dedicated to encoding functions, which constitute the more technical part of our demonstration.

6.1 Preliminaries

Assume we are given some set of formulas Φ, referred to as *kernel*, which is free of set quantifiers and closed under negation (e.g., $\overrightarrow{\text{ML}}_{\text{g}}$). Then, for $\ell \geqslant 0$, the class $\Sigma_\ell^{\text{MSO}}(\Phi)$ consists of those formulas that one can construct by taking a member of Φ and prepending to it at most ℓ consecutive blocks of set quantifiers, alternating between existential and universal blocks, such that the first block is existential. Reformulating this solely in terms of existential quantifiers and negations, we get

$$\begin{aligned}&\Sigma_0^{\text{MSO}}(\Phi) \coloneqq \Phi \quad \text{and}\\ &\Sigma_{\ell+1}^{\text{MSO}}(\Phi) \coloneqq \{\exists_X \mid X \in \mathbb{S}_1\}^* \cdot \{\neg\varphi \mid \varphi \in \Sigma_\ell^{\text{MSO}}(\Phi)\},\end{aligned}$$

where the second line uses set concatenation and the Kleene star. We define $\Pi_\ell^{\text{MSO}}(\Phi)$ as the corresponding dual class, i.e., the set of all negations of formulas in $\Sigma_\ell^{\text{MSO}}(\Phi)$. Generalizing this to arbitrary Boolean combinations, let $\text{BC}\,\Sigma_\ell^{\text{MSO}}(\Phi)$ denote the smallest superclass of $\Sigma_\ell^{\text{MSO}}(\Phi)$ that is closed under negation and disjunction.

The formulas in $\Sigma_\ell^{\text{MSO}}(\Phi)$ and $\Pi_\ell^{\text{MSO}}(\Phi)$ are said to be in *prenex normal form* with respect to the kernel Φ. It is well known that every MSOL-formula can be transformed into prenex normal form with kernel class FOL. This is based on the observation that first-order quantifiers can be replaced by second-order ones. Using the construction of Example 2.2 in Section 2.6, it is not difficult to see that the analogue holds for $\text{MSO}(\overrightarrow{\text{ML}})$, $\text{MSO}(\overleftrightarrow{\text{ML}})$, $\text{MSO}(\overrightarrow{\text{ML}}_{\text{g}})$ and $\text{MSO}(\overleftrightarrow{\text{ML}}_{\text{g}})$ with respect to their corresponding kernel classes. A more elaborate explanation can be found in [Cat06, Prp. 3].

For the sake of clarity, we break with the tradition of implicit quantification that is customary in modal logic. Instead of evaluating $\text{MSO}(\overrightarrow{\text{ML}})$-formulas on non-pointed structures by means of "hidden" universal quantification, we shall explicitly put a global box in front of our formulas. This leads to the class

$$\boxdot\,\Sigma_\ell^{\text{MSO}}(\overrightarrow{\text{ML}}) \coloneqq \{\boxdot\} \cdot \Sigma_\ell^{\text{MSO}}(\overrightarrow{\text{ML}}).$$

Analogously, we also define $\boxdot\,\Pi_\ell^{\text{MSO}}(\overrightarrow{\text{ML}})$.

All of our results will be stated in terms of the semantic classes that one obtains by evaluating the preceding formula classes on some set of structures $\mathcal{C}$. On the semantic side, we will additionally consider the class

$$[\![\Delta_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \coloneqq [\![\Sigma_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \cap [\![\Pi_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}}.$$

Since it is not based on any syntactic counterpart, there is no meaning attributed to the notation $\Delta_\ell^{\text{MSO}}(\Phi)$ by itself (without the brackets).

6.2 Separation results

With the notation in place, we are ready to formally enunciate the main theorem, whose complete proof will be the subject of the remainder of this chapter. It is an

Separation result	*Kernel* Class Φ	*Structures* Class $\mathcal{C}$	*Levels* $\ell \geqslant \cdot$	*Theorem*
$[\![\Delta^{\text{MSO}}_{\ell+1}(\Phi)]\!]_{\mathcal{C}} \nsubseteq [\![\text{BC}\,\Sigma^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}}$	FOL	GRID, DG, GRAPH	1	6.1 (a) *
	$\overleftrightarrow{\text{ML}}_g$, $\overrightarrow{\text{ML}}_g$	GRID, DG, GRAPH$^1_1$	1	6.2 (a)
$[\![\Sigma^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}} \overset{\nsubseteq}{\nsupseteq} [\![\Pi^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}}$	FOL	GRID, DG, GRAPH	1	6.1 (b) *
	$\overleftrightarrow{\text{ML}}_g$, $\overrightarrow{\text{ML}}_g$	GRID, DG, GRAPH$^1_1$	1	6.2 (b)
	$\overrightarrow{\text{ML}}$	@DG	1	6.2 (c)
$[\![\boxdot\,\Sigma^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}} \nsubseteq [\![\boxdot\,\Pi^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}}$	$\overrightarrow{\text{ML}}$	DG	2	6.2 (d)

Table 6.1. The specific separation results of Theorems 6.1 and 6.2. Theorem 6.1 (marked by asterisks) is due to Matz, Schweikardt and Thomas.

extension to modal kernel formulas of the following result of Matz, Schweikardt and Thomas, obtained by combining [MST02, Thm. 1] and [Mat02, Thm. 2.26][1]:

Theorem 6.1 (Matz, Schweikardt, Thomas).
▶ The set quantifier alternation hierarchy of MSOL is strict over the classes of grids, digraphs and undirected graphs.
A more precise statement of this theorem, referred to as Theorem 6.1 (a) and (b), is given in Table 6.1. ◀

Roughly speaking, the extension provided in the present chapter tells us that the preceding separations are largely maintained if we replace the first-order kernel by certain classes of modal formulas. To facilitate comparisons, the formal statements of both theorems are presented together in the same table.

Theorem 6.2 (Main Results).
▶ The set quantifier alternation hierarchies of $\text{MSO}(\overleftrightarrow{\text{ML}}_g)$ and $\text{MSO}(\overrightarrow{\text{ML}}_g)$ are strict over the classes of grids, digraphs and 1-bit labeled undirected graphs.
Furthermore, the corresponding hierarchies of $\text{MSO}(\overrightarrow{\text{ML}})$ and $\boxdot\,\text{MSO}(\overrightarrow{\text{ML}})$ are (mostly) strict over the classes of pointed digraphs and digraphs, respectively.
A more precise statement of this theorem, referred to as Theorem 6.2 (a), (b), (c) and (d), is given in Table 6.1. ◀

In particular, the inclusion of $[\![\text{BC}\,\Sigma^{\text{MSO}}_{\ell}(\Phi)]\!]_{\mathcal{C}}$ in $[\![\Delta^{\text{MSO}}_{\ell+1}(\Phi)]\!]_{\mathcal{C}}$ follows from the fact that, when transforming a Boolean combination of $\Sigma^{\text{MSO}}_{\ell}(\Phi)$-formulas into prenex normal form, one is free to choose whether the resulting formula (with up to $\ell+1$ quantifier alternations) should start with an existential or a universal quantifier.

By basic properties of predicate logic and the transitivity of set inclusion, it is easy to infer from Theorem 6.2 the hierarchy diagrams represented in Figures 6.1 and 6.2.
If we take into account all the depicted relations, the diagram in Figure 6.1 is the same as in [MST02] and [Mat02]. Hence, when switching to one of the modal kernels that include global modalities, i.e., $\overleftrightarrow{\text{ML}}_g$ or $\overrightarrow{\text{ML}}_g$, the separations of Theorem 6.1 are completely preserved on grids and digraphs. Our proof method also allows us to easily transfer this result to undirected graphs, as long as we admit that the vertices may be labeled with at least one bit. Additional work would be required to eliminate this condition.

[1] [Mat02, Thm. 2.26] states that $[\![\Sigma^{\text{MSO}}_{\ell}(\text{FOL})]\!]_{\text{GRID}} \nsupseteq [\![\Pi^{\text{MSO}}_{\ell}(\text{FOL})]\!]_{\text{GRID}}$, which, by duality, also implies $[\![\Sigma^{\text{MSO}}_{\ell}(\text{FOL})]\!]_{\text{GRID}} \nsubseteq [\![\Pi^{\text{MSO}}_{\ell}(\text{FOL})]\!]_{\text{GRID}}$.

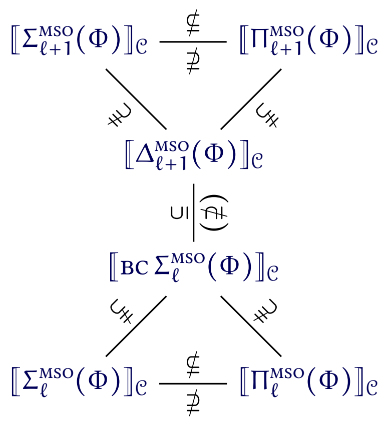

Figure 6.1. The set quantifier alternation hierarchies established by Theorem 6.2 (a), (b) and (c). If we include the noninclusion in parentheses, this diagram holds for $\Phi \in \{\overleftrightarrow{\text{ML}}_{\text{g}}, \overrightarrow{\text{ML}}_{\text{g}}\}$ and $\mathcal{C} \in \{\text{GRID}, \text{DG}, \text{GRAPH}_1^1\}$. If we ignore that noninclusion, it is also verified for $\Phi = \overrightarrow{\text{ML}}$ and $\mathcal{C} = @\text{DG}$. In both cases, we assume $\ell \geqslant 1$.

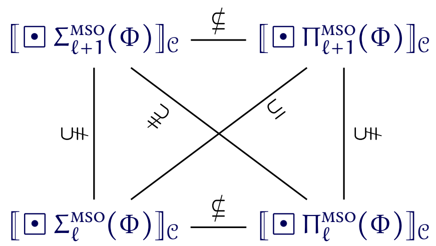

Figure 6.2. The set quantifier alternation hierarchy implied by Theorem 6.2 (d) for $\Phi = \overrightarrow{\text{ML}}$, $\mathcal{C} = \text{DG}$, and $\ell \geqslant 2$.

As a spin-off, Theorem 6.2 also provides an extension of some of these separations to $\overrightarrow{\text{ML}}$, a kernel class without global modalities. Following [Kuu08, Kuu15], we consider the alternation hierarchies of both $\text{MSO}(\overrightarrow{\text{ML}})$ and $\boxdot\,\text{MSO}(\overrightarrow{\text{ML}})$. For the former, which is evaluated on pointed digraphs, Figure 6.1 gives a detailed picture, leaving open only whether the inclusion $[\![\text{BC}\,\Sigma_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \subseteq [\![\Delta_{\ell+1}^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}}$ is proper. Inferring the strictness of this inclusion from the preceding results does not seem very difficult, but would call for a generalization of our framework. In contrast, the second hierarchy based on $\overrightarrow{\text{ML}}$ is arguably less natural, since every $\boxdot\,\text{MSO}(\overrightarrow{\text{ML}})$-formula is prefixed by a global box, regardless of the occurring set quantifiers. This creates a certain asymmetry between the Σ_ℓ^{MSO}- and Π_ℓ^{MSO}-levels, which becomes apparent when considering the missing relations in Figure 6.2. Unlike for the other hierarchies, one cannot simply argue by duality to deduce from $[\![\boxdot\,\Sigma_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \nsubseteq [\![\boxdot\,\Pi_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}}$ that the converse noninclusion also holds. Nevertheless, the presented result is strong enough to answer the specific strictness question mentioned in [Kuu08]: For arbitrarily high ℓ, we have

$$[\![\boxdot\,\Sigma_\ell^{\text{MSO}}(\overrightarrow{\text{ML}})]\!]_{\text{DG}} \nsupseteq [\![\boxdot\,\Sigma_{\ell+1}^{\text{MSO}}(\overrightarrow{\text{ML}})]\!]_{\text{DG}}.$$

6.3 Top-level proofs

In accordance with our top-down approach, the present section already provides the proof of our main theorem, where everything comes together. It therefore acts as a gateway to the sections with the technical parts, especially Section 6.5.

6.3.1 Figurative inclusions

First of all, we need to introduce the primary tool with which we will transfer separation results from one setting to another. It can be seen as an abstraction of the *strong first-order reductions* used in [MST02]. Unlike the latter, it is formulated independently of any logical language, which allows us to postpone the technical details to the end of the chapter.

Definition 6.3 (Figurative Inclusion).
▸ Consider two sets $\mathcal{C}$ and $\mathcal{D}$ and a partial *injective* function $\mu\colon \mathcal{C} \rightharpoonup \mathcal{D}$. For any two families of subsets $\mathcal{L} \subseteq 2^{\mathcal{C}}$ and $\mathcal{M} \subseteq 2^{\mathcal{D}}$, we say that $\mathcal{L}$ is *forward included* in $\mathcal{M}$ *figuro* μ, and write $\mathcal{L} \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M}$, if for every set $L \in \mathcal{L}$, there is a set $M \in \mathcal{M}$ such that $\mu(L) = M \cap \mu(\mathcal{C})$. ◂

Figuratively speaking, the partial bijection μ creates a tunnel between $\mathcal{C}$ and $\mathcal{D}$, and all the sets in $\mathcal{L}$ and $\mathcal{M}$ are cropped to fit through that tunnel. Two original sets are considered to be equal if their cropped versions are mapped onto each other by μ.

We denote the inverse function of μ by μ^{-1} and the identity function on $\mathcal{C}$ by $\mathrm{id}_{\mathcal{C}}$.

We also define the shorthands $\vec{\supseteq}_{\mu}$ and $\vec{\equiv}_{\mu}$ as natural extensions of the previous notation: $\mathcal{L} \mathrel{\vec{\supseteq}_{\mu}} \mathcal{M}$, which is defined as $\mathcal{M} \mathrel{\vec{\subseteq}_{\mu^{-1}}} \mathcal{L}$, means that $\mathcal{M}$ is *backward included* in $\mathcal{L}$ *figuro* μ, and $\mathcal{L} \mathrel{\vec{\equiv}_{\mu}} \mathcal{M}$, an abbreviation for the conjunction of $\mathcal{L} \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M}$ and $\mathcal{L} \mathrel{\vec{\supseteq}_{\mu}} \mathcal{M}$, states that $\mathcal{L}$ is *forward equal* to $\mathcal{M}$ *figuro* μ. All of these relations are referred to as *figurative inclusions*.

Note that ordinary inclusion is a special case of figurative inclusion, i.e., for $\mathcal{C} = \mathcal{D}$,

$$\mathcal{L} \subseteq \mathcal{M} \quad \text{if and only if} \quad \mathcal{L} \mathrel{\vec{\subseteq}_{\mathrm{id}_{\mathcal{C}}}} \mathcal{M}.$$

Furthermore, figurative inclusion is transitive in the sense that

$$\mathcal{L} \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M} \mathrel{\vec{\subseteq}_{\nu}} \mathcal{N} \quad \text{implies} \quad \mathcal{L} \mathrel{\vec{\subseteq}_{\nu\circ\mu}} \mathcal{N}.$$

(This depends crucially on the fact that ν is injective.)

Proof. Consider three sets $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$, two partial injective functions $\mu\colon \mathcal{C} \rightharpoonup \mathcal{D}$ and $\nu\colon \mathcal{D} \rightharpoonup \mathcal{E}$, and three families of subsets $\mathcal{L} \subseteq 2^{\mathcal{C}}$, $\mathcal{M} \subseteq 2^{\mathcal{D}}$ and $\mathcal{N} \subseteq 2^{\mathcal{E}}$. Assume that we have $\mathcal{L} \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M} \mathrel{\vec{\subseteq}_{\nu}} \mathcal{N}$. Choose an arbitrary set $L \in \mathcal{L}$. Since $\mathcal{L} \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M}$, there must be a set $M \in \mathcal{M}$ such that $\mu(L) = M \cap \mu(\mathcal{C})$. Furthermore, as $\mathcal{M} \mathrel{\vec{\subseteq}_{\nu}} \mathcal{N}$, there is also a set $N \in \mathcal{N}$ such that $\nu(M) = N \cap \nu(\mathcal{D})$. Hence,

$$\begin{aligned}
(\nu \circ \mu)(L) &= \nu(M \cap \mu(\mathcal{C})) \\
&= \nu(M) \cap (\nu \circ \mu)(\mathcal{C}) && (*) \\
&= N \cap \nu(\mathcal{D}) \cap (\nu \circ \mu)(\mathcal{C}) \\
&= N \cap (\nu \circ \mu)(\mathcal{C}).
\end{aligned}$$

Equality $(*)$ holds because ν is injective. Since the choice of L was arbitrary, there is such an $N \in \mathcal{N}$ for every $L \in \mathcal{L}$, and thus $\mathcal{L} \mathrel{\vec{\subseteq}_{\nu\circ\mu}} \mathcal{N}$. ∎

In our specific context, given a noninclusion $[\![\Phi_2]\!]_{\mathcal{C}} \nsubseteq [\![\Phi_1]\!]_{\mathcal{C}}$, we shall use the concept of figurative inclusion to infer from it another noninclusion $[\![\Psi_2]\!]_{\mathcal{D}} \nsubseteq [\![\Psi_1]\!]_{\mathcal{D}}$. Here, $\Phi_1, \Phi_2, \Psi_1, \Psi_2$ and $\mathcal{C}, \mathcal{D}$ refer to some classes of formulas and structures, respectively. The key part of the argument will be to construct an appropriate encoding function $\mu\colon \mathcal{C} \to \mathcal{D}$, in order to apply the following lemma.

Lemma 6.4 .
▸ Let $\mathcal{L}_1, \mathcal{L}_2 \subseteq 2^{\mathcal{C}}$ and $\mathcal{M}_1, \mathcal{M}_2 \subseteq 2^{\mathcal{D}}$ be families of subsets of some sets $\mathcal{C}$ and $\mathcal{D}$. If there is a *total injective* function $\mu\colon \mathcal{C} \to \mathcal{D}$ such that $\mathcal{L}_2 \mathrel{\vec{\subseteq}_{\mu}} \mathcal{M}_2$ and $\mathcal{L}_1 \mathrel{\vec{\supseteq}_{\mu}} \mathcal{M}_1$, then

$$\mathcal{L}_2 \nsubseteq \mathcal{L}_1 \quad \text{implies} \quad \mathcal{M}_2 \nsubseteq \mathcal{M}_1. \quad ◂$$

Proof. To show the contrapositive, let us suppose that $\mathcal{M}_2 \subseteq \mathcal{M}_1$, or, equivalently, $\mathcal{M}_2 \mathrel{\vec{\sqsubseteq}}_{\mathrm{id}_{\mathcal{D}}} \mathcal{M}_1$. Then the chain of figurative inclusions

$$\mathcal{L}_2 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_2 \mathrel{\vec{\sqsubseteq}}_{\mathrm{id}_{\mathcal{D}}} \mathcal{M}_1 \mathrel{\vec{\sqsubseteq}}_{\mu^{-1}} \mathcal{L}_1$$

yields $\mathcal{L}_2 \mathrel{\vec{\sqsubseteq}}_{\mathrm{id}_{\mathcal{C}}} \mathcal{L}_1$, since $(\mu^{-1} \circ \mathrm{id}_{\mathcal{D}} \circ \mu) = \mathrm{id}_{\mathcal{C}}$. (This depends on μ being total and injective.) Consequently, we have $\mathcal{L}_2 \subseteq \mathcal{L}_1$. ∎

In some cases, we can combine two given figurative inclusions in order to obtain a new one that relates the corresponding intersection classes. This property will be very useful for establishing figurative inclusions between classes of the form $[\![\Delta_\ell^{\mathrm{MSO}}(\Phi)]\!]_{\mathcal{C}}$.

Lemma 6.5 .
▶ Consider two sets $\mathcal{C}$ and $\mathcal{D}$, a partial injective function $\mu\colon \mathcal{C} \rightharpoonup \mathcal{D}$, and four families of subsets $\mathcal{L}_1, \mathcal{L}_2 \subseteq 2^{\mathcal{C}}$ and $\mathcal{M}_1, \mathcal{M}_2 \subseteq 2^{\mathcal{D}}$. If $\mu(\mathcal{C})$ is a member of $\mathcal{M}_1 \cap \mathcal{M}_2$, and $\mathcal{M}_1$, $\mathcal{M}_2$ are both closed under intersection, then

$$\mathcal{L}_1 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_1 \quad \text{and} \quad \mathcal{L}_2 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_2 \qquad \text{imply} \quad \mathcal{L}_1 \cap \mathcal{L}_2 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_1 \cap \mathcal{M}_2. \quad ◀$$

Proof. Let L be any set in $\mathcal{L}_1 \cap \mathcal{L}_2$. Since $\mathcal{L}_1 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_1$, there is, by definition, a set M in $\mathcal{M}_1$ such that $\mu(L) = M \cap \mu(\mathcal{C})$. Furthermore, we also know that $\mu(\mathcal{C})$ lies in $\mathcal{M}_1$, and that the latter is closed under intersection. Hence, $\mu(L) \in \mathcal{M}_1$. Analogously, we also get that $\mu(L) \in \mathcal{M}_2$. Finally, knowing that for all L in $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mu(L)$ lies in $\mathcal{M}_1 \cap \mathcal{M}_2$, we obviously have a sufficient condition for $\mathcal{L}_1 \cap \mathcal{L}_2 \mathrel{\vec{\sqsubseteq}}_{\mu} \mathcal{M}_1 \cap \mathcal{M}_2$. ∎

6.3.2 Proving the main theorem

We are now ready to give the central proof of this chapter. Although it makes references to many statements of Sections 6.4 and 6.5, it is formulated in a way that can be understood without having read anything beyond this point.

Proof of Theorem 6.2. The basis of our proof shall be laid in Section 6.4, where the case $s = 0$ of Theorem 6.9 will state the following: When restricted to the class of grids, the set quantifier alternation hierarchies of MSOL, $\mathrm{MSO}(\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}})$ and $\mathrm{MSO}(\vec{\mathrm{ML}}_{\mathrm{g}})$ are equivalent. More precisely, for every $\ell \geqslant 1$ and $\Xi \in \{\Sigma_\ell^{\mathrm{MSO}}, \Pi_\ell^{\mathrm{MSO}}, \mathrm{BC}\,\Sigma_\ell^{\mathrm{MSO}}, \Delta_\ell^{\mathrm{MSO}}\}$, it holds that

$$[\![\Xi(\mathrm{FOL})]\!]_{\mathrm{GRID}} = [\![\Xi(\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{GRID}} = [\![\Xi(\vec{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{GRID}}.$$

Hence, if we consider only the case $\mathcal{C} = \mathrm{GRID}$, the separation results for the kernel class FOL stated in Theorem 6.1 (a) and (b) immediately imply those for $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$ and $\vec{\mathrm{ML}}_{\mathrm{g}}$ in Theorem 6.2 (a) and (b).

The remainder of the proof now consists of establishing suitable figurative inclusions, in order to transfer these results to other classes of structures and, to some extent, to weaker classes of kernel formulas. For this purpose, we shall introduce in Section 6.5 a notion of translatability between two classes of kernel formulas Φ and Ψ, with respect to a given total injective function μ that encodes structures from a class $\mathcal{C}$ into structures of some class $\mathcal{D}$. As will be shown in Lemma 6.14, bidirectional translatability implies

$$[\![\Xi(\Phi)]\!]_{\mathcal{C}} \mathrel{\vec{\equiv}}_{\mu} [\![\Xi(\Psi)]\!]_{\mathcal{D}} \tag{$*$}$$

for all $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}\}$ with $\ell \geqslant 0$. If we can additionally show that $\mu(\mathcal{C})$ is (at most) $\Delta_2^{\text{MSO}}(\Psi)$-definable over $\mathcal{D}$, then, by Lemma 6.5, the figurative equality $(*)$ also holds for $\Xi = \Delta_{\ell+1}^{\text{MSO}}$ with $\ell \geqslant 1$. Note that the backward part "$\overleftarrow{\sqsupseteq}_\mu$" is always true, since $\mu^{-1}(\mathcal{D})$ is trivially $\Delta_2^{\text{MSO}}(\Phi)$-definable over $\mathcal{C}$.

The groundwork being in place, we proceed by applying Lemma 6.4 as follows:

- If we have established $(*)$ for $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}\}$, then we can transfer the separation
$$[\![\Sigma_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \underset{\not\supseteq}{\not\subseteq} [\![\Pi_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \tag{1}$$
to the kernel class Ψ evaluated on the class of structures $\mathcal{D}$.
- Similarly, if $(*)$ holds for $\Xi \in \{\text{BC}\,\Sigma_\ell^{\text{MSO}}, \Delta_{\ell+1}^{\text{MSO}}\}$, then
$$[\![\Delta_{\ell+1}^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \not\subseteq [\![\text{BC}\,\Sigma_\ell^{\text{MSO}}(\Phi)]\!]_{\mathcal{C}} \tag{2}$$
can also be transferred to Ψ on $\mathcal{D}$.

It remains to provide concrete figurative inclusions to prove the different parts of Theorem 6.2.

(a), (b) The first two parts are treated in parallel. We start by transferring (1) and (2) from grids to digraphs, for the kernel class $\overleftrightarrow{\text{ML}}_{\text{g}}$, taking a detour via 2-relational digraphs, and then via 2-bit labeled ones. For all $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}, \Delta_{\ell+1}^{\text{MSO}}\}$ with $\ell \geqslant 1$, we get

$$\begin{aligned}
[\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{GRID}} &\overleftrightarrow{\sqsupseteq}_{\text{id}_{\text{GRID}}} [\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}_0^2} \\
&\overleftrightarrow{\sqsupseteq}_{\mu_1} \quad [\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}_2^1} \\
&\overleftrightarrow{\sqsupseteq}_{\mu_2} \quad [\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}}.
\end{aligned}$$

The first line is trivial for $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}\}$, since $\text{GRID} \subseteq \text{DG}_0^2$. It also holds for $\Xi = \Delta_{\ell+1}^{\text{MSO}}$ because GRID is $\Pi_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_{\text{g}})$-definable over DG_0^2, as shall be demonstrated in Proposition 6.10. The other two lines rely on the existence of adequate injective functions μ_1 and μ_2 that allow us to apply Lemmas 6.14 and 6.5 in the way explained above. They will be provided by Propositions 6.15 and 6.16, respectively.

We proceed in a similar way to transfer (1) and (2) from $\overleftrightarrow{\text{ML}}_{\text{g}}$ to $\overrightarrow{\text{ML}}_{\text{g}}$ on digraphs:

$$\begin{aligned}
[\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}} &\overleftrightarrow{\sqsupseteq}_{\mu_3} [\![\Xi(\overrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}_0^2} \\
&\overleftrightarrow{\sqsupseteq}_{\mu_1} [\![\Xi(\overrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}_2^1} \\
&\overleftrightarrow{\sqsupseteq}_{\mu_2} [\![\Xi(\overrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}},
\end{aligned}$$

for $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}, \Delta_{\ell+1}^{\text{MSO}}\}$ with $\ell \geqslant 1$. The very simple encoding function μ_3, which lets us eliminate backward modalities and again use Lemmas 6.14 and 6.5, will be supplied by Proposition 6.17. The encodings μ_1 and μ_2 are the same as before, because the properties asserted by Propositions 6.15 and 6.16 hold for both $\overrightarrow{\text{ML}}_{\text{g}}$ and $\overleftrightarrow{\text{ML}}_{\text{g}}$ as kernel classes. Incidentally, this means we could transfer (1) directly from $\overrightarrow{\text{ML}}_{\text{g}}$ on grids to $\overrightarrow{\text{ML}}_{\text{g}}$ on digraphs, without even mentioning $\overleftrightarrow{\text{ML}}_{\text{g}}$.

To show that (1) and (2) are also valid for $\overrightarrow{\text{ML}}_{\text{g}}$ on 1-bit labeled undirected graphs, we establish

$$[\![\Xi(\overleftrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{DG}} \overleftrightarrow{\sqsupseteq}_{\mu_4} [\![\Xi(\overrightarrow{\text{ML}}_{\text{g}})]\!]_{\text{GRAPH}_1^1},$$

again for all $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}, \Delta_{\ell+1}^{\text{MSO}}\}$ with $\ell \geqslant 1$. The appropriate encoding μ_4 shall be constructed in Proposition 6.18. Since backward modalities do not offer any additional expressive power on undirected graphs, the separations we obtain also hold for the kernel $\overleftrightarrow{\text{ML}}_{\text{g}}$.

(c) Next, to transfer (1) from $\overrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on digraphs to $\overrightarrow{\mathrm{ML}}$ on pointed digraphs, we show that, for $\Xi \in \{\Sigma_\ell^{\mathrm{MSO}}, \Pi_\ell^{\mathrm{MSO}}\}$ with $\ell \geqslant 1$, we have

$$[\![\Xi(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{DG}} \overset{\rightarrow}{=}_{\mu_5} [\![\Xi(\overrightarrow{\mathrm{ML}})]\!]_{@\mathrm{DG}}.$$

The injective function μ_5, which satisfies the translatability property required to obtain this figurative equality via Lemma 6.14, will be provided by Proposition 6.19. Its image $\mu_5(\mathrm{DG})$ is not $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$-definable, for the simple reason that an $\mathrm{MSO}(\overrightarrow{\mathrm{ML}})$-formula is unable to distinguish between two structures that are isomorphic when restricted to the connected component containing the position marker @. Hence, we cannot merely apply Lemma 6.5 to show (2). Our approach would have to be refined to take into account equivalence classes of structures, which we shall not do in this thesis.

(d) Finally, Proposition 6.19 will also state that μ_5 can be converted into an encoding μ_5', from DG back into DG, that satisfies the following figurative inclusions for all $\ell \geqslant 2$:

$$[\![\Sigma_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{DG}} \overset{\rightarrow}{\subseteq}_{\mu_5'} [\![\boxdot\, \Sigma_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}})]\!]_{\mathrm{DG}},$$
$$[\![\Pi_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{DG}} \overset{\rightarrow}{=}_{\mu_5'} [\![\boxdot\, \Pi_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}})]\!]_{\mathrm{DG}}.$$

Using (1) for $\overrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on digraphs, and applying Lemma 6.4, we can infer from this that

$$[\![\boxdot\, \Sigma_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}})]\!]_{\mathrm{DG}} \nsubseteq [\![\boxdot\, \Pi_\ell^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}})]\!]_{\mathrm{DG}}. \qquad \blacksquare$$

6.4 Grids as a starting point

In this section, we establish that the set quantifier alternation hierarchies of MSOL, $\mathrm{MSO}(\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}})$ and $\mathrm{MSO}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$ are equivalent on labeled grids. In addition, we give a $[\Pi_1^{\mathrm{MSO}}(\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}})]$-formula that characterizes the class of grids.

6.4.1 The standard translation

Our first building block is a well-known property of modal logic, which holds even if we do not confine ourselves to the setting of grids.

Proposition 6.6 .
▸ For every $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$-formula, there is an equivalent FOL-formula, i.e.,

$$[\![\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}]\!] \subseteq [\![\mathrm{FOL}]\!]. \qquad ◂$$

Proof. Given an $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$-formula φ, we have to construct an FOL-formula ψ_φ such that $G \vDash \varphi$ if and only if $G \vDash \psi_\varphi$, for every structure G. This is simply a matter of transcribing the semantics of $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$ given in Table 2.1 to the language of first-order logic, a method known as the *standard translation* in modal logic (see, e.g., [BRV02, Def. 2.45]). The following table gives a recursive specification of this translation.

$\varphi \in \overleftrightarrow{\mathrm{ML}}_g$	Equivalent formula $\psi_\varphi \in$ FOL
x	$@ \doteq x$
X	$X(@)$
$\neg\varphi_1$	$\neg\psi_{\varphi_1}$
$\varphi_1 \vee \varphi_2$	$\psi_{\varphi_1} \vee \psi_{\varphi_2}$
$\langle R \rangle(\varphi_1, \dots, \varphi_k)$	$\exists_{x_1,\dots,x_k}\big(R(@, x_1, \dots, x_k) \;\wedge\; \bigwedge_{1 \leqslant i \leqslant k} \psi_{\varphi_i}[@ \mapsto x_i]\big)$
$\langle \bar{R} \rangle(\varphi_1, \dots, \varphi_k)$	as above, except $R(x_k, \dots, x_1, @)$
$\langle\cdot\rangle\varphi_1$	$\exists_@ \psi_{\varphi_1}$

Here, $x \in \mathbb{S}_0$, $X \in \mathbb{S}_1$, $R \in \mathbb{S}_{k+1}$, $\varphi_1, \dots, \varphi_k \in \overleftrightarrow{\mathrm{ML}}_g$, for $k \geqslant 1$, and $x_1, \dots, x_k$ are node symbols, chosen such that $x_i \notin \mathrm{free}(\psi_{\varphi_i})$. The notation $\psi_{\varphi_i}[@ \mapsto x_i]$ designates the formula obtained by substituting each free occurrence of $@$ in ψ_{φ_i} by x_i. ∎

6.4.2 A detour through tiling systems

By restricting our focus to the class of labeled grids, we can take advantage of a well-studied automaton model introduced by Giammarresi and Restivo in [GR92], which is closely related to MSOL. A "machine" in this model, called a *tiling system*, is defined as a tuple $T = (\Sigma, Q, \Theta)$, where

- $\Sigma = \mathbb{2}^s$ is seen as an alphabet, with $s \geqslant 0$,
- Q is a finite set of sates, and
- $\Theta \subseteq \big((\Sigma \times Q) \cup \{\#\}\big)^4$ is a set of 2×2-tiles that may use a fresh letter # not contained in $(\Sigma \times Q)$.

For a fixed number of bits s, we denote by TS_s the set of all tiling systems with alphabet $\Sigma = \mathbb{2}^s$.

Given a s-bit labeled grid G, a tiling system $T \in \mathrm{TS}_s$ operates similarly to a nondeterministic finite automaton generalized to two dimensions. A run of T on G is an extended labeled grid $G^\#$, obtained by nondeterministically labeling each cell of G with some state $q \in Q$ and surrounding the entire grid with a border consisting of new #-labeled cells. We consider $G^\#$ to be a valid run if each of its 2×2-subgrids can be identified with some tile in Θ. The set recognized by T consists precisely of those labeled grids for which such a run exists. By analogy with our existing notation, we write $[\![\mathrm{TS}_s]\!]_{\mathrm{GRID}_s}$ for the class formed by the sets of s-bit labeled grids that are recognized by some tiling system in TS_s.

Exploiting a locality property of first-order logic, Giammarresi, Restivo, Seibert and Thomas have shown in [GRST96] that tiling systems capture precisely the existential fragment of MSOL on labeled grids:

Theorem 6.7 (Giammarresi, Restivo, Seibert, Thomas).
▶ For arbitrary $s \geqslant 0$, a set of s-bit labeled grids is TS-recognizable if and only if it is $\Sigma_1^{\mathrm{MSO}}(\mathrm{FOL})$-definable over GRID_s, i.e.,

$$[\![\mathrm{TS}_s]\!]_{\mathrm{GRID}_s} = [\![\Sigma_1^{\mathrm{MSO}}(\mathrm{FOL})]\!]_{\mathrm{GRID}_s}.$$ ◀

The preceding result is extremely useful for our purposes, because, from the perspective of modal logic, it provides a much easier access to MSOL. This brings us to the following proposition.

Proposition 6.8 .
▶ For arbitrary $s \geqslant 0$, if a set of s-bit labeled grids is TS-recognizable, then it is also $\Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g)$-definable over GRID_s, i.e.,

$$[\![\text{TS}_s]\!]_{\text{GRID}_s} \subseteq [\![\Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g)]\!]_{\text{GRID}_s}. \quad ◀$$

Proof. Let $T = (\Sigma, Q, \Theta)$ be a tiling system with alphabet $\Sigma = \mathbb{2}^s$. We have to construct a $\Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g)$-sentence φ_T over the signature $\{P_1, \dots, P_s, R_1, R_2\}$, such that each labeled grid $G \in \text{GRID}_s$ satisfies φ_T if and only if it is accepted by T.

The idea is standard: We represent the states of T by additional set symbols $(X_q)_{q \in Q}$, and our formula asserts that there exists a corresponding partition of V^G into $|Q|$ subsets that represent a run $G^\#$ of T on G. To verify that it is indeed a valid run, we have to check that each 2×2-subgrid of $G^\#$ corresponds to some tile

$$\theta = \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_3 & \theta_4 \end{bmatrix}$$

in Θ. If the entry θ_1 is different from #, we can easily write down an $\overrightarrow{\text{ML}}$-formula φ_θ that checks at a given position $v \in V^G$, whether the 2×2-subgrid of $G^\#$ with upper-left corner v matches θ. Here, θ_1 is chosen as the representative entry of θ, because the upper-left corner of the tile can “see” the other nodes by following the directed R_1- and R_2-edges. Otherwise, if θ_1 is equal to #, there is no such node v, since G does not contain special border nodes. However, we can always choose some other entry θ_i, different from #, to be the representative of θ, and write a formula φ_θ describing the tile from the point of view of a node corresponding to θ_i. This choice is never arbitrary, because the representative must be able to “see” the other non-# entries of the tile. Consequently, we divide Θ into four disjoint sets Θ_1, Θ_2, Θ_3, Θ_4, such that Θ_i contains those tiles θ that are represented by their entry θ_i. In order to facilitate the subsequent formalization, we further subdivide each set into partitions according to the #-borders that occur within the tiles: Θ_{M} contains the “middle tiles” (all entries different from #), Θ_{L} the “left tiles” (with θ_1 and θ_3 equal to #), Θ_{BR} the “bottom-right tiles”, and so forth … Altogether, Θ is partitioned into nine subsets, grouped into four types:

$$\Theta_1 = \Theta_{\text{M}} \mathbin{\dot\cup} \Theta_{\text{B}} \mathbin{\dot\cup} \Theta_{\text{R}} \mathbin{\dot\cup} \Theta_{\text{BR}} \qquad \Theta_2 = \Theta_{\text{L}} \mathbin{\dot\cup} \Theta_{\text{BL}}$$
$$\Theta_3 = \Theta_{\text{T}} \mathbin{\dot\cup} \Theta_{\text{TR}} \qquad \Theta_4 = \Theta_{\text{TL}}$$

We now construct the formula φ_T in a bottom-up manner, starting with a subformula φ_{θ_i} for each entry θ_i other than #, for every tile $\theta \in \Theta$. Letting θ_i be equal to $(a, q) \in \Sigma \times Q$, with $a = a_1 \dots a_s$, the formula φ_{θ_i} checks at a given position $v \in V^G$ if the labeling of v matches θ_i.

$$\varphi_{\theta_i} = \bigwedge_{a_j = 1} P_j \wedge \bigwedge_{a_j = 0} \neg P_j \wedge X_q \wedge \bigwedge_{q' \neq q} \neg X_{q'}$$

Building on this, we can define for each tile $\theta \in \Theta$ the formula φ_θ mentioned above. Since $\overrightarrow{\text{ML}}_g$ does not have backward modalities, there is a certain asymmetry between tiles in Θ_1, where the representative can "see" the entire 2×2-subgrid, and the remaining tiles, where the representative must "know" that it lies in the leftmost column or the uppermost row of the grid G. We shall address this issue shortly, and just assume that information not accessible to the representative is verified by another part of the ultimate formula φ_T. For tiles in $\Theta_M, \Theta_{BR}, \Theta_L, \Theta_{TL}$, the definitions of φ_θ are given in the following table. For tiles in $\Theta_B, \Theta_R, \Theta_{BL}, \Theta_T, \Theta_{TR}$, the method is completely analogous.

θ	φ_θ
$\Theta_M \ni \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_3 & \theta_4 \end{bmatrix}$	$\varphi_{\theta_1} \wedge \langle 2 \rangle \varphi_{\theta_2} \wedge \langle 1 \rangle \varphi_{\theta_3} \wedge \langle 1 \rangle \langle 2 \rangle \varphi_{\theta_4}$
$\Theta_{BR} \ni \begin{bmatrix} \theta_1 & \# \\ \# & \# \end{bmatrix}$	$\varphi_{\theta_1} \wedge [2]\bot \wedge [1]\bot$
$\Theta_L \ni \begin{bmatrix} \# & \theta_2 \\ \# & \theta_4 \end{bmatrix}$	$\varphi_{\theta_2} \wedge \langle 1 \rangle \varphi_{\theta_4}$
$\Theta_{TL} \ni \begin{bmatrix} \# & \# \\ \# & \theta_4 \end{bmatrix}$	φ_{θ_4}

It remains to mark the top and left borders of G, using two additional predicates Y_T and Y_L, over which we will quantify existentially. To this end, we write an $\overrightarrow{\text{ML}}_g$-formula φ_{border}, checking that top [resp. left] nodes have no R_1- [resp. R_2-] predecessor, that there is a top-left node, and that being top [resp. left] is passed on to the R_2- [resp. R_1-] successor, if it exists.

$$\begin{aligned}\varphi_{\text{border}} = {} & \neg \langle\cdot\rangle \big(\langle \bar{1} \rangle Y_T \vee \langle \bar{2} \rangle Y_L \big) \;\wedge\; \langle\cdot\rangle \big(Y_T \wedge Y_L \big) \;\wedge \\ & [\cdot] \Big(\big(Y_T \rightarrow [2] Y_T \big) \wedge \big(Y_L \rightarrow [1] Y_L \big) \Big)\end{aligned}$$

Finally, we can put everything together to describe the acceptance condition of T. Every node $v \in V^G$ has to ensure that it corresponds to the upper-left corner of some tile in Θ_1. Furthermore, nodes in the leftmost column or uppermost row of G must additionally check that the assignment of states is compatible with the tiles in Θ_2, Θ_3, Θ_4. This leads to the desired formula φ_T:

$$\begin{aligned}\exists_{(X_q)_{q \in Q}, Y_T, Y_L} \Bigg(& \varphi_{\text{border}} \;\wedge \\ & [\cdot] \Big(\bigvee_{\theta \in \Theta_1} \varphi_\theta \Big) \;\wedge\; [\cdot] \Big(Y_L \rightarrow \bigvee_{\theta \in \Theta_2} \varphi_\theta \Big) \;\wedge \\ & [\cdot] \Big(Y_T \rightarrow \bigvee_{\theta \in \Theta_3} \varphi_\theta \Big) \;\wedge\; [\cdot] \Big(Y_T \wedge Y_L \rightarrow \bigvee_{\theta \in \Theta_4} \varphi_\theta \Big) \Bigg)\end{aligned}$$

Note that we do not need a separate subformula to check that the interpretations of $(X_q)_{q \in Q}$ form a partition of V^G, since this is already done implicitly in the conjunct $[\cdot]\big(\bigvee_{\theta \in \Theta_1} \varphi_\theta\big)$. ■

6.4.3 Equivalent hierarchies on grids

We now have all we need to prove the levelwise equivalence of MSOL, $\text{MSO}(\overleftrightarrow{\text{ML}}_g)$ and $\text{MSO}(\overrightarrow{\text{ML}}_g)$ on labeled grids.

Theorem 6.9 .

▶ Let $s \geqslant 0$, $\ell \geqslant 1$ and $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}, \text{BC}\,\Sigma_\ell^{\text{MSO}}, \Delta_\ell^{\text{MSO}}\}$. When restricted to the class of s-bit labeled grids, $\Xi(\text{FOL})$, $\Xi(\overleftrightarrow{\text{ML}}_g)$ and $\Xi(\overrightarrow{\text{ML}}_g)$ are equivalent, i.e.,

$$\begin{aligned}\llbracket \Xi(\text{FOL}) \rrbracket_{\text{GRID}_s} &= \llbracket \Xi(\overleftrightarrow{\text{ML}}_g) \rrbracket_{\text{GRID}_s} \\ &= \llbracket \Xi(\overrightarrow{\text{ML}}_g) \rrbracket_{\text{GRID}_s}.\end{aligned}$$ ◀

Proof. First, we show that the claim holds for the case $\Xi = \Sigma_1^{\text{MSO}}$ (with arbitrary $s \geqslant 0$). This can be seen from the following circular chain of inclusions:

$$\begin{aligned}\llbracket \Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g) \rrbracket_{\text{GRID}_s} &\subseteq \llbracket \Sigma_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g) \rrbracket_{\text{GRID}_s} &\text{(a)}\\ &\subseteq \llbracket \Sigma_1^{\text{MSO}}(\text{FOL}) \rrbracket_{\text{GRID}_s} &\text{(b)}\\ &\subseteq \llbracket \text{TS}_s \rrbracket_{\text{GRID}_s} &\text{(c)}\\ &\subseteq \llbracket \Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g) \rrbracket_{\text{GRID}_s} &\text{(d)}\end{aligned}$$

(a) The first inclusion follows from the fact that $\Sigma_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g)$ is a syntactic fragment of $\Sigma_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g)$.

(b) For the second inclusion, consider any $\Sigma_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g)$-formula $\widehat{\varphi} = \exists_{X_1,\dots,X_n}(\varphi)$, where $X_1, \dots, X_n$ are set symbols and φ is an $\overleftrightarrow{\text{ML}}_g$-formula. By Proposition 6.6, we can replace φ in $\widehat{\varphi}$ by an equivalent FOL-formula ψ_φ. This results in the $\Sigma_1^{\text{MSO}}(\text{FOL})$-formula $\psi_{\widehat{\varphi}} = \exists_{X_1,\dots,X_n}(\psi_\varphi)$, which is equivalent to $\widehat{\varphi}$ on arbitrary structures, and thus, in particular, on s-bit labeled grids.

(c) The translation from $\Sigma_1^{\text{MSO}}(\text{FOL})$ on labeled grids to tiling systems corresponds to the more challenging direction of Theorem 6.7, which is the main result of [GRST96].

(d) The last inclusion is given by Proposition 6.8.

The general version of the theorem can now be obtained by induction on ℓ. This is straightforward, because the classes $\Pi_\ell^{\text{MSO}}(\Phi)$, $\text{BC}\,\Sigma_\ell^{\text{MSO}}(\Phi)$ and $\Sigma_{\ell+1}^{\text{MSO}}(\Phi)$ are defined syntactically in terms of $\Sigma_\ell^{\text{MSO}}(\Phi)$, for any set of kernel formulas Φ (see Section 6.1), and if the claim holds for $\Xi \in \{\Sigma_\ell^{\text{MSO}}, \Pi_\ell^{\text{MSO}}\}$, then it also holds for the intersection classes of the form $\llbracket \Delta_\ell^{\text{MSO}}(\Phi) \rrbracket_{\text{GRID}_s}$. ■

6.4.4 A logical characterization of grids

We conclude this section by showing that a single layer of universal set quantifiers is enough to describe grids in $\text{MSO}(\overleftrightarrow{\text{ML}}_g)$.

Proposition 6.10 .

▶ The set of all grids is $\Pi_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g)$-definable over 2-relational digraphs, i.e.,

$$\text{GRID} \in \llbracket \Pi_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g) \rrbracket_{\text{DG}_0^2}.$$ ◀

Proof. In the course of this proof, we give a list of properties, items a to f, which are obviously necessary for a 2-relational digraph G to be a grid, and show how to express them as $[\Pi_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g)]$-formulas. We argue that the conjunction of all of these properties also constitutes a sufficient condition for being a grid, which immediately provides us with the required formula, since $\llbracket \Pi_1^{\text{MSO}}(\overleftrightarrow{\text{ML}}_g) \rrbracket$ is closed under intersection.

a. For each relation symbol $R \in \{R_1, R_2\}$, every node has at most one R-predecessor and at most one R-successor; in other words, R_1^G and R_2^G are partial injective functions.

$$\bigwedge_{R \in \{R_1, R_1^{-1}, R_2, R_2^{-1}\}} \forall X\, \boxdot\big(\langle R\rangle X \to [R] X\big)$$

b. Again considering each $R \in \{R_1, R_2\}$ separately, there is a directed R-path from every node to an R-sink, i.e., to some node without R-successor.

$$\bigwedge_{R \in \{R_1, R_2\}} \forall X\Big(\langle\cdot\rangle X \wedge \boxdot\big(X \to [R] X\big) \to \langle\cdot\rangle\big(X \wedge [R]\bot\big)\Big)$$

Taken together, properties a and b state that R_1^G and R_2^G each form a collection of directed, acyclic, pairwise vertex-disjoint paths. Let us refer to the first nodes of those paths as R_1- and R_2-sources, respectively.

c. There is precisely one node that is both an R_1- and an R_2-source.

$$\mathrm{tot1}\big([\overline{1}]\bot \wedge [\overline{2}]\bot\big)$$

(Here, tot1 is the schema from Example 2.2 in Section 2.6.)

d. The R_1-predecessors and R_1-successors of R_2-sources must be R_2-sources themselves.

$$\boxdot\big([\overline{2}]\bot \to [\overline{1}][\overline{2}]\bot \wedge [1][\overline{2}]\bot\big)$$

By adding c and d to our list of conditions, we ensure that there is an R_1-path consisting precisely of the R_2-sources, thereby also forcing the digraph G to be connected.

e. If a node has both an R_1- and an R_2-successor, then it also has a descendant reachable by first taking an R_1-edge and then an R_2-edge.

$$\boxdot\big(\langle 1\rangle\top \wedge \langle 2\rangle\top \to \langle 1\rangle\langle 2\rangle\top\big)$$

f. The relations R_1^G and R_2^G commute. This means that following an R_1-edge and then an R_2-edge leads to the same node as first taking an R_2-edge and then an R_1-edge.

$$\forall X\, \boxdot\big(\langle 1\rangle\langle 2\rangle X \leftrightarrow \langle 2\rangle\langle 1\rangle X\big)$$

Considered in conjunction with condition a, there are only two ways to satisfy e and f from the point of view of two nodes $u, v \in V^G$ that are connected by an R_1-edge from u to v: either both nodes are R_2-sinks, or they have R_2-successors u' and v', respectively, with an R_1-edge from u' to v'. Moreover, v' only possesses an R_1-successor if v does. Now, imagine we start from the left border, i.e., from the R_1-path that consists of all the R_2-sources, which is provided by properties a to d, and iteratively enforce the requirements just mentioned. Then, in doing so, we propagate the grid topology through the entire digraph. More specifically, the additional requirements of e and f entail that all the R_2-paths have the same length, and that the nodes lying at a fixed (horizontal) position of those R_2-paths constitute an independent R_1-path, ordered in the same way as their respective R_2-predecessors. ∎

6.5 A toolbox of encodings

In this section, we provide all the encoding functions used in the proof of Theorem 6.2 (see Section 6.3.2), and show that they satisfy suitable translatability properties, allowing us to establish the required figurative inclusions. With a view to modularity and reusability, some of our constructions are more general than needed.

Given a set of symbols σ, the extension $\sigma \cup \{@\}$ will be abbreviated to $\sigma_@$.

6.5.1 Encodings that allow for translation

We shall only consider encoding functions that are linear in the following sense:

Definition 6.11 (Linear Encoding).
▶ Let $\mathcal{C}$, $\mathcal{D}$ be two classes of structures, and m, n be integers such that $1 \leqslant m \leqslant n$. A *linear encoding* from $\mathcal{C}$ into $\mathcal{D}$ with parameters m, n is a total injective function $\mu\colon \mathcal{C} \to \mathcal{D}$ that assigns to each structure $G \in \mathcal{C}$ a structure $\mu(G) \in \mathcal{D}$, whose domain is composed of m disjoint copies of the domain of G and $n - m$ additional nodes, i.e.,

$$V^{\mu(G)} = \big([1:m] \times V^G\big) \;\cup\;]m:n]. \qquad ◀$$

Given such a linear encoding μ and some $\overleftrightarrow{\mathrm{ML}}_g$-formula φ, we want to be able to construct a new formula ψ_φ, such that evaluating φ on $\mathcal{C}$ is equivalent to evaluating ψ_φ on $\mu(\mathcal{C})$. Conversely, we also desire a way of constructing a formula φ_ψ that is equivalent on $\mathcal{C}$ to a given formula ψ on $\mu(\mathcal{C})$. The following two definitions formalize this translatability property for both directions. We then show in Lemma 6.14 that they adequately capture our intended meaning. Although the underlying idea is very simple, the presentation is a bit lengthy because we have to exhaustively cover the structure of $\overleftrightarrow{\mathrm{ML}}_g$-formulas.

Definition 6.12 (Forward Translation).
▶ Consider two classes of structures $\mathcal{C}$ and $\mathcal{D}$ over signatures σ and τ, respectively, two classes of formulas $\Phi, \Psi \in \{\overrightarrow{\mathrm{ML}}, \overleftrightarrow{\mathrm{ML}}, \overrightarrow{\mathrm{ML}}_g, \overleftrightarrow{\mathrm{ML}}_g\}$, and a linear encoding $\mu\colon \mathcal{C} \to \mathcal{D}$. We say that μ allows for *forward translation* from Φ to Ψ if the following properties are satisfied:

a. For each node symbol or set symbol P in σ, there is a Ψ-sentence ψ_P over $\tau_@$, such that

$$G[@ \mapsto u] \models P \quad \text{iff} \quad \mu(G)[@ \mapsto (1,u)] \models \psi_P,$$

for all $G \in \mathcal{C}$ and $u \in V^G$.

b. For each relation symbol R in σ of arity $k + 1 \geqslant 2$, there is a Ψ-sentence ψ_R over $\tau_@$ enriched with additional set symbols $(Y_i)_{1 \leqslant i \leqslant k}$, such that

$$G\big[@, (X_i)_{i \leqslant k} \mapsto u, (U_i)_{i \leqslant k}\big] \models \langle\!\langle R \rangle\!\rangle (X_i)_{i \leqslant k}$$
if and only if
$$\mu(G)\big[@, (Y_i)_{i \leqslant k} \mapsto (1,u), (W_i)_{i \leqslant k}\big] \models \psi_R,$$
assuming U_i, W_i satisfy $u' \in U_i \Leftrightarrow (1,u') \in W_i$,

for all $G \in \mathcal{C}$, $u \in V^G$, sets $(U_i)_{1 \leqslant i \leqslant k} \subseteq V^G$ and $(W_i)_{1 \leqslant i \leqslant k} \subseteq V^{\mu(G)}$, and set symbols $(X_i)_{1 \leqslant i \leqslant k}$.

c. If Φ includes backward modalities, then for each relation symbol R in σ of arity at least 2, there is a Ψ-formula $\psi_{R^{-1}}$ that satisfies the property of item b for R^{-1} instead of R.

d. If Φ includes global modalities, then there is a Ψ-formula $\psi_{\bullet}$ that satisfies the property of item b for $\bullet$ instead of R and $k = 1$.

e. There is a Ψ-sentence ψ_{ini} over τ enriched with an additional set symbol Y, such that

$$G[X \mapsto U] \models \frac{X}{\langle\!\cdot\!\rangle X} \quad \text{iff} \quad \mu(G)[Y \mapsto W] \models \psi_{\mathrm{ini}},$$

assuming U, W satisfy $u \in U \Leftrightarrow (1, u) \in W$,
where $\frac{X}{\langle\!\cdot\!\rangle X}$ is X if $@ \in \sigma$, and $\langle\!\cdot\!\rangle X$ otherwise,

for all $G \in \mathcal{C}$, $U \subseteq V^G$, $W \subseteq V^{\mu(G)}$ and $X \in \mathbb{S}_1$. ◀

Definition 6.13 (Backward Translation).
▶ Consider two classes of structures $\mathcal{C}$ and $\mathcal{D}$ over signatures σ and τ, respectively, two classes of formulas $\Phi, \Psi \in \{\overrightarrow{\mathrm{ML}}, \overleftrightarrow{\mathrm{ML}}, \overrightarrow{\mathrm{ML}}_g, \overleftrightarrow{\mathrm{ML}}_g\}$, and a linear encoding $\mu: \mathcal{C} \to \mathcal{D}$ with parameters m, n. We say that μ allows for *backward translation* from Ψ to Φ if the following properties are satisfied:

a. For each node symbol or set symbol Q in τ and all $h \in [n]$, there is a Φ-sentence φ^h_Q over $\sigma_@$, such that

$$G[@ \mapsto u] \models \varphi^h_Q \quad \text{iff} \quad \mu(G)[@ \mapsto v] \models Q,$$

where v is (h, u) if $h \leqslant m$, and h otherwise,

for all $G \in \mathcal{C}$ and $u \in V^G$.

b. For each relation symbol S in τ of arity $k + 1 \geqslant 2$, and all $h \in [n]$, there is a Φ-sentence φ^h_S over $\sigma_@$ enriched with additional set symbols $(X^j_i)^{1 \leqslant j \leqslant n}_{1 \leqslant i \leqslant k}$, such that

$$G\big[@, (X^j_i)^{j \leqslant n}_{i \leqslant k} \mapsto u, (U^j_i)^{j \leqslant n}_{i \leqslant k}\big] \models \varphi^h_S$$

if and only if

$$\mu(G)\big[@, (Y_i)_{i \leqslant k} \mapsto v, (W_i)_{i \leqslant k}\big] \models \langle S \rangle (Y_i)_{i \leqslant k},$$

where v is (h, u) if $h \leqslant m$, otherwise h, and

$$W_i = \bigcup_{1 \leqslant j \leqslant m} \big(\{j\} \times U^j_i\big) \cup \bigcup_{m < j \leqslant n} \{j \mid U^j_i = V^G\},$$

for all $G \in \mathcal{C}$, nodes $u \in V^G$, sets $(U^j_i)^{1 \leqslant j \leqslant m}_{1 \leqslant i \leqslant k} \subseteq V^G$ and $(U^j_i)^{m < j \leqslant n}_{1 \leqslant i \leqslant k} \in \{\varnothing, V^G\}$, and set symbols $(Y_i)_{1 \leqslant i \leqslant k}$.

c. If Ψ includes backward modalities, then for each relation symbol S in τ of arity at least 2, and all $h \in [n]$, there is a Φ-formula $\varphi^h_{S^{-1}}$ that satisfies the property of item b for S^{-1} instead of S.

d. If Ψ includes global modalities, then for all $h \in [n]$, there is a Φ-formula $\varphi^h_{\bullet}$ that satisfies the property of item b for $\bullet$ instead of S and $k = 1$.

e. There is a Φ-sentence φ_{ini} over σ enriched with additional set symbols $(X^j)^{1\leqslant j\leqslant n}$, such that

$$G\big[(X^j)^{j\leqslant n} \mapsto (U^j)^{j\leqslant n}\big] \models \varphi_{\mathrm{ini}}$$

if and only if

$$\mu(G)[Y \mapsto W] \models \frac{Y}{\diamondsuit Y},$$

where $\frac{Y}{\diamondsuit Y}$ is Y if $@ \in \tau$, otherwise $\diamondsuit Y$, and

$$W = \bigcup_{1\leqslant j\leqslant m}\big(\{j\}\times U^j\big) \cup \bigcup_{m<j\leqslant n}\{j \mid U^j = V^G\},$$

for all structures $G \in \mathcal{C}$, sets $(U^j)^{1\leqslant j\leqslant m} \subseteq V^G$ and $(U^j)^{m<j\leqslant n} \in \{\varnothing, V^G\}$, and $Y \in \mathbb{S}_1$. ◂

To simplify matters slightly, we shall say that a linear encoding μ allows for *bidirectional translation* between Φ and Ψ, if it allows for both forward translation from Φ to Ψ and backward translation from Ψ to Φ. Furthermore, in case $\Phi = \Psi$, we may say "within Φ" instead of "between Φ and Φ".

Let us now prove that our notion of translatability is indeed sufficient to imply figurative inclusion on the semantic side, even if we bring set quantifiers into play.

Lemma 6.14 .

▸ Consider two classes of structures $\mathcal{C}$ and $\mathcal{D}$, a linear encoding $\mu\colon \mathcal{C} \to \mathcal{D}$, two classes of formulas $\Phi, \Psi \in \{\overrightarrow{\mathrm{ML}}, \overleftrightarrow{\mathrm{ML}}, \overrightarrow{\mathrm{ML}}_g, \overleftrightarrow{\mathrm{ML}}_g\}$, and let $\Xi \in \{\Sigma_\ell^{\mathrm{MSO}}, \Pi_\ell^{\mathrm{MSO}}, \mathrm{BC}\,\Sigma_\ell^{\mathrm{MSO}}\}$, for some arbitrary $\ell \geqslant 0$.

a. If μ allows for forward translation from Φ to Ψ, then we have

$$[\![\Xi(\Phi)]\!]_{\mathcal{C}} \;\vec{\subseteq}_\mu\; [\![\Xi(\Psi)]\!]_{\mathcal{D}}.$$

b. Similarly, if μ allows for backward translation from Ψ to Φ, then we have

$$[\![\Xi(\Phi)]\!]_{\mathcal{C}} \;\vec{\supseteq}_\mu\; [\![\Xi(\Psi)]\!]_{\mathcal{D}}.$$ ◂

Proof. Let σ and τ be the signatures underlying $\mathcal{C}$ and $\mathcal{D}$, respectively. Parts a and b of the lemma are treated separately in the following proof.

In several places, given some $\mathrm{MSO}(\overleftrightarrow{\mathrm{ML}}_g)$-formula φ, the need will arise to substitute newly created $\overleftrightarrow{\mathrm{ML}}_g$-formulas $\varphi_1, \dots, \varphi_k$ for set symbols $X_1, \dots, X_k$. We shall write $\varphi[(X_i)_{i\leqslant k} \mapsto (\varphi_i)_{i\leqslant k}]$ to denote the $\mathrm{MSO}(\overleftrightarrow{\mathrm{ML}}_g)$-formula that one obtains by simultaneously replacing every free occurrence of each X_i in φ by the formula φ_i.

a. For every $\Xi(\Phi)$-sentence φ over σ, we must construct a $\Xi(\Psi)$-sentence ψ_φ over τ, such that ψ_φ says about $\mu(G)$ the same as φ says about G, for all structures $G \in \mathcal{C}$.

We start by focusing on the kernel classes Φ, Ψ, and show the following by induction on the structure of Φ-formulas: For every Φ-sentence φ over $\sigma_@ \cup \mathcal{Z}$, with $\mathcal{Z} = \{Z_1, \dots, Z_z\}$ being any collection of set symbols disjoint from σ and τ (i.e., free set variables), there is a Ψ-sentence ψ_φ^* over $\tau_@ \cup \mathcal{Z}$ such that

$$G\big[@, (Z_t)_{t\leqslant z} \mapsto u, (U_t)_{t\leqslant z}\big] \models \varphi$$

if and only if

$$\mu(G)\big[@, (Z_t)_{t\leqslant z} \mapsto (1,u), (W_t)_{t\leqslant z}\big] \models \psi_\varphi^*,$$

assuming U_t, W_t satisfy $u' \in U_t \Leftrightarrow (1,u') \in W_t$, for all structures $G \in \mathcal{C}$, nodes $u \in V^G$, and sets $(U_t)_{1 \leqslant t \leqslant z} \subseteq V^G$ and $(W_t)_{1 \leqslant t \leqslant z} \subseteq V^{\mu(G)}$.

- If $\varphi = @$ or $\varphi = Z$, for some $Z \in \mathcal{Z}$, it suffices to set $\psi^*_\varphi = \varphi$.

- If $\varphi = P$, for some node symbol or set symbol P in σ, we exploit that μ allows for forward translation from Φ to Ψ, and choose $\psi^*_\varphi = \psi_P$. Here, ψ_P is the formula postulated by Definition 6.12 a; it fulfills the induction hypothesis, since adding interpretations of the symbols $Z_1, \dots, Z_z$ to a structure has no influence on whether or not that structure satisfies a sentence over a signature that does not contain these symbols.

- If $\varphi = \neg\varphi_1$ or $\varphi = \varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are formulas that satisfy the induction hypothesis, we set $\psi^*_\varphi = \neg\psi^*_{\varphi_1}$ or $\psi^*_\varphi = \psi^*_{\varphi_1} \vee \psi^*_{\varphi_2}$, respectively.

- If $\varphi = \langle R \rangle(\varphi_i)_{i \leqslant k}$, where R is a relation symbol in σ of arity $k+1 \geqslant 2$, and $(\varphi_i)_{i \leqslant k}$ are Φ-sentences over $\sigma_@ \cup \mathcal{Z}$ satisfying the induction hypothesis, we again use the fact that μ allows for forward translation from Φ to Ψ. The desired formula ψ^*_φ is obtained by substituting $(\psi^*_{\varphi_i})_{i \leqslant k}$ for the symbols $(Y_i)_{i \leqslant k}$ in the formula ψ_R, whose existence is asserted by Definition 6.12 b, i.e.,

$$\psi^*_\varphi = \psi_R\big[(Y_i)_{i \leqslant k} \mapsto (\psi^*_{\varphi_i})_{i \leqslant k}\big].$$

  For any integer $i \in [1:k]$, let U'_i be the set of nodes $u' \in V^G$ that satisfy φ_i in $G\big[(Z_t)_{t \leqslant z} \mapsto (U_t)_{t \leqslant z}\big]$, and let W'_i be the set of nodes $v' \in V^{\mu(G)}$ that satisfy ψ_{φ_i} in $\mu(G)\big[(Z_t)_{t \leqslant z} \mapsto (W_t)_{t \leqslant z}\big]$. By induction hypothesis, we are guaranteed that all the sets U'_i, W'_i are such that a node u' lies in U'_i if and only if $(1, u')$ lies in W'_i. Thus, we have

$$\begin{aligned} & G\big[@, (Z_t)_{t \leqslant z} \mapsto u, (U_t)_{t \leqslant z}\big] \models \varphi \\ \text{iff} \quad & G\big[@, (X_i)_{i \leqslant k} \mapsto u, (U'_i)_{i \leqslant k}\big] \models \langle R \rangle(X_i)_{i \leqslant k} \\ \text{iff} \quad & \mu(G)\big[@, (Y_i)_{i \leqslant k} \mapsto (1, u), (W'_i)_{i \leqslant k}\big] \models \psi_R \\ \text{iff} \quad & \mu(G)\big[@, (Z_t)_{t \leqslant z} \mapsto (1, u), (W_t)_{t \leqslant z}\big] \models \psi^*_\varphi. \end{aligned}$$

- If $\varphi = \langle \overline{R} \rangle(\varphi_i)_{i \leqslant k}$, assuming Φ incorporates backward modalities, we obtain ψ^*_φ by applying the same argument as in the previous case, but this time considering R^{-1} instead of R and invoking Definition 6.12 c.

- If $\varphi = \langle \bullet \rangle\varphi_1$, supposing Φ includes global modalities, we again follow the same line of reasoning as in the case $\varphi = \langle R \rangle(\varphi_i)_{i \leqslant k}$, referring to Definition 6.12 d and using $\bullet$ instead of R, with $k = 1$.

Now we can consider the case where the position symbol $@$ is not (re)mapped, and then look beyond the kernel classes to finally deal with set quantifiers. Arguing once more by structural induction, we extend the preceding claim as follows: For every $\Xi(\Phi)$-sentence φ over $\sigma \cup \mathcal{Z}$, with $\mathcal{Z} = \{Z_1, \dots, Z_z\}$ as before (possibly empty), there is a $\Xi(\Psi)$-sentence ψ_φ over $\tau \cup \mathcal{Z}$ such that

$$G\big[(Z_t)_{t \leqslant z} \mapsto (U_t)_{t \leqslant z}\big] \models \varphi$$

if and only if

$$\mu(G)\big[(Z_t)_{t \leqslant z} \mapsto (W_t)_{t \leqslant z}\big] \models \psi_\varphi,$$

assuming U_t, W_t satisfy $u \in U_t \Leftrightarrow (1, u) \in W_t$, for all $G \in \mathcal{C}$, $(U_t)_{1 \leqslant t \leqslant z} \subseteq V^G$ and $(W_t)_{1 \leqslant t \leqslant z} \subseteq V^{\mu(G)}$.

- If φ lies in the kernel Φ, we make use of the claim just proven, together with the formula ψ_{ini} described in Definition 6.12 e. We set $\psi_\varphi = \psi_{\text{ini}}[Y \mapsto \psi^*_\varphi]$.
  - If @ belongs to σ, the asserted property of ψ_{ini} guarantees that φ holds at the initial position $@^G$ in the $\mathcal{Z}$-extended variant of G if and only if ψ_φ is satisfied by the $\mathcal{Z}$-extended variant of $\mu(G)$.
  - Otherwise, @ cannot be free in φ, since φ is a sentence over $\sigma \cup \mathcal{Z}$, which also implies that Φ incorporates global modalities. It follows that φ is equivalent to $\diamond\varphi$. Again applying the definition of ψ_{ini}, we obtain that the $\mathcal{Z}$-extended variant of G satisfies $\diamond\varphi$, and thus φ, if and only if the $\mathcal{Z}$-extended variant of $\mu(G)$ satisfies ψ_φ.

- If φ is a Boolean combination of formulas that satisfy the induction hypothesis, the translation is straightforward, just as in the previous part of the proof.

- If $\varphi = \exists_{Z_{z+1}} \varphi_1$, where φ_1 is a $\Xi(\Phi)$-sentence over $\sigma \cup \{Z_1, \dots, Z_{z+1}\}$ that satisfies the hypothesis, we choose $\psi_\varphi = \exists_{Z_{z+1}} \psi_{\varphi_1}$. To justify this choice, let G' and $\mu(G)'$ denote the $\mathcal{Z}$-extended variants of G and $\mu(G)$, respectively. We get the following by induction:
  - If choosing $Z_{z+1} \mapsto U_{z+1}$ leads to satisfaction of φ_1 in G', then choosing $Z_{z+1} \mapsto \{1\} \times U_{z+1}$ does the same for ψ_{φ_1} in $\mu(G)'$.
  - Conversely, if $Z_{z+1} \mapsto W_{z+1}$ is a satisfying choice for ψ_{φ_1} in $\mu(G)'$, then so is $Z_{z+1} \mapsto \{u \mid (1, u) \in W_{z+1}\}$ for φ_1 in G'.

b. The proof of the reverse direction of the lemma is very similar to the previous one, but a bit more cumbersome, because each node of a structure G has to play the role of several different nodes in $\mu(G)$. Given any $\Xi(\Psi)$-sentence ψ over τ, we need to construct a $\Xi(\Phi)$-sentence φ_ψ over σ, such that evaluating φ_ψ on G is equivalent to evaluating ψ on $\mu(G)$, for all $G \in \mathcal{C}$. For the remainder of this proof, let m, n be the parameters of the linear encoding μ.

Again, we first deal with the kernel classes Φ, Ψ, and show the following claim by induction on the structure of Ψ-formulas: For every Ψ-sentence ψ over $\tau_@ \cup \mathcal{Z}$ and all $h \in [n]$, with $\mathcal{Z} = \{Z_1, \dots, Z_z\} \subseteq \mathbb{S}_1 \setminus \tau$, there is a Φ-sentence φ^h_ψ over $\sigma_@ \cup \tilde{\mathcal{Z}}$, with $\tilde{\mathcal{Z}} = \{Z^1_1, \dots, Z^n_z\} \subseteq \mathbb{S}_1 \setminus \sigma$, such that

$$G\big[@, (Z^j_t)^{j \leqslant n}_{t \leqslant z} \mapsto u, (U^j_t)^{j \leqslant n}_{t \leqslant z}\big] \models \varphi^h_\psi$$

if and only if

$$\mu(G)\big[@, (Z_t)_{t \leqslant z} \mapsto v, (W_t)_{t \leqslant z}\big] \models \psi,$$

where v is (h, u) if $h \leqslant m$, otherwise h, and

$$W_t = \bigcup_{1 \leqslant j \leqslant m} \big(\{j\} \times U^j_t\big) \cup \bigcup_{m < j \leqslant n} \{j \mid U^j_t = V^G\},$$

for all $G \in \mathcal{C}$, $u \in V^G$, and sets $(U^j_t)^{1 \leqslant j \leqslant m}_{1 \leqslant t \leqslant z} \subseteq V^G$ and $(U^j_t)^{m < j \leqslant n}_{1 \leqslant t \leqslant z} \in \{\varnothing, V^G\}$.

- If $\psi = @$, it suffices to set $\varphi^h_\psi = @$.

- If $\psi = Z_t$, for some $Z_t \in \mathcal{Z}$, the translation is given by $\varphi^h_\psi = Z^h_t$.

- If $\psi = Q$, for some node symbol or set symbol Q in τ, we use the fact that μ allows for backward translation from Ψ to Φ, and choose φ^h_ψ to be the formula φ^h_Q, which is provided by Definition 6.13 a. The definition asserts that this formula fulfills the induction hypothesis for the case where G and $\mu(G)$ are not extended using additional set symbols from $\tilde{\mathcal{Z}}$ and $\mathcal{Z}$. But since these symbols do not occur freely in φ^h_Q and Q, their interpretations do not influence the evaluation of the formulas.

- If $\psi = \neg\psi_1$ or $\psi = \psi_1 \vee \psi_2$, where ψ_1 and ψ_2 are formulas that satisfy the induction hypothesis, we set $\varphi^h_\psi = \neg\varphi^h_{\psi_1}$ or $\varphi^h_\psi = \varphi^h_{\psi_1} \vee \varphi^h_{\psi_2}$, respectively.

- If $\psi = \Diamond_S(\psi_i)_{i\leqslant k}$, where S is a relation symbol in τ of arity $k{+}1 \geqslant 2$, and $(\psi_i)_{i\leqslant k}$ are Ψ-sentences over $\tau_@ \cup \mathcal{Z}$ satisfying the hypothesis, we again rely on the premise that μ allows for backward translation from Ψ to Φ. We construct φ^h_ψ by plugging the formulas $(\varphi^j_{\psi_i})^{j\leqslant n}_{i\leqslant k}$ provided by induction into the formula φ^h_S of Definition 6.13 b as follows:

$$\varphi^h_\psi = \varphi^h_S\big[(X^j_i)^{j\leqslant n}_{i\leqslant k} \mapsto (\varphi^j_{\psi_i})^{j\leqslant n}_{i\leqslant k}\big].$$

  For $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n$, let $U^{j\prime}_i$ be the set of nodes $u' \in V^G$ that satisfy $\varphi^j_{\psi_i}$ in $G\big[(Z^j_t)^{j\leqslant n}_{t\leqslant z} \mapsto (U^j_t)^{j\leqslant n}_{t\leqslant z}\big]$, and let W'_i be the set of nodes $v' \in V^{\mu(G)}$ that satisfy ψ_i in $\mu(G)\big[(Z_t)_{t\leqslant z} \mapsto (W_t)_{t\leqslant z}\big]$. The induction hypothesis ensures that

$$W'_i = \bigcup_{1\leqslant j\leqslant m}\big(\{j\}{\times}U^{j\prime}_i\big) \cup \bigcup_{m<j\leqslant n}\{j \mid U^{j\prime}_i = V^G\}.$$

  Hence, we obtain the required equivalence as follows:

$$\begin{aligned}
&G\big[@, (Z^j_t)^{j\leqslant n}_{t\leqslant z} \mapsto u, (U^j_t)^{j\leqslant n}_{t\leqslant z}\big] \vDash \varphi^h_\psi \\
\text{iff}\quad &G\big[@, (X^j_i)^{j\leqslant n}_{i\leqslant k} \mapsto u, (U^{j\prime}_i)^{j\leqslant n}_{i\leqslant k}\big] \vDash \varphi^h_S \\
\text{iff}\quad &\mu(G)\big[@, (Y_i)_{i\leqslant k} \mapsto v, (W'_i)_{i\leqslant k}\big] \vDash \Diamond_S(Y_i)_{i\leqslant k} \\
\text{iff}\quad &\mu(G)\big[@, (Z_t)_{t\leqslant z} \mapsto v, (W_t)_{t\leqslant z}\big] \vDash \psi.
\end{aligned}$$

- If $\psi = \overline{\Diamond}_S(\psi_i)_{i\leqslant k}$, supposing Ψ includes backward modalities, we construct φ^h_ψ using the same approach as in the previous case, the only difference being that we consider S^{-1} instead of S and invoke Definition 6.13 c instead of 6.13 b.

- If $\psi = \Diamond_\bullet\psi_1$, in case Ψ includes global modalities, we again proceed as for the case $\psi = \Diamond_S(\psi_i)_{i\leqslant k}$, this time using $\bullet$ instead of S, with $k = 1$, and referring to Definition 6.13 d.

Similarly to the proof of part a, we now extend the previous property to cover formulas with set quantifiers, evaluated on structures that may interpret the position symbol @ arbitrarily. Our induction hypothesis is the following: For every $\Xi(\Psi)$-sentence ψ over $\tau \cup \mathcal{Z}$, with $\mathcal{Z} = \{Z_1, \dots, Z_z\} \subseteq \mathbb{S}_1 \setminus \tau$ (possibly empty), there is a $\Xi(\Phi)$-sentence φ_ψ over $\sigma \cup \tilde{\mathcal{Z}}$, with $\tilde{\mathcal{Z}} = \{Z^1_1, \dots, Z^n_z\} \subseteq \mathbb{S}_1 \setminus \sigma$, such that

$$\begin{gathered}
G\big[(Z^j_t)^{j\leqslant n}_{t\leqslant z} \mapsto (U^j_t)^{j\leqslant n}_{t\leqslant z}\big] \vDash \varphi_\psi \\
\text{if and only if} \\
\mu(G)\big[(Z_t)_{t\leqslant z} \mapsto (W_t)_{t\leqslant z}\big] \vDash \psi, \quad \text{where} \\
W_t = \bigcup_{1\leqslant j\leqslant m}\big(\{j\}{\times}U^j_t\big) \cup \bigcup_{m<j\leqslant n}\{j \mid U^j_t = V^G\},
\end{gathered}$$

for all structures $G \in \mathcal{C}$, and sets $(U_t^j)_{1\leqslant t\leqslant z}^{1\leqslant j\leqslant m} \subseteq V^G$ and $(U_t^j)_{1\leqslant t\leqslant z}^{m<j\leqslant n} \in \{\varnothing, V^G\}$.

- If ψ belongs to the kernel class Ψ, we apply the claim just proven, and construct φ_ψ by substituting into the formula φ_{ini} provided by Definition 6.13 e: $\varphi_\psi = \varphi_{\mathrm{ini}}\big[(X^j)^{j\leqslant n} \mapsto (\varphi_\psi^j)^{j\leqslant n}\big]$. Proceeding analogously to the proof of part a, we have to distinguish whether or not the position symbol @ belongs to τ. (If it does not, ψ is necessarily equivalent to $\langle\!\cdot\!\rangle\psi$.) In both cases, the definition of φ_{ini} guarantees that the $\tilde{\mathcal{Z}}$-extended variant of G satisfies φ_ψ if and only if the $\mathcal{Z}$-extended variant of $\mu(G)$ satisfies ψ.

- If ψ is a Boolean combination of subformulas that satisfy the induction hypothesis, then φ_ψ is simply the corresponding Boolean combination of the translated subformulas.

- If $\psi = \exists_{Z_{z+1}}\psi_1$, where ψ_1 is a $\Xi(\Psi)$-sentence over $\tau \cup \{Z_1, \dots, Z_{z+1}\}$ that satisfies the induction hypothesis, we choose φ_ψ to be the formula

$$\exists_{(Z_{z+1}^j)^{j\leqslant m}}\Big(\bigvee_{N\subseteq\,]m:n]} \varphi_{\psi_1}\big[(Z_{z+1}^j)^{j>m} \mapsto (N(j))^{j>m}\big]\Big),$$

with $N(j) = \top$ if $j \in N$, and $N(j) = \bot$ otherwise. For each set $N \subseteq\,]m:n]$, let $\varphi_{\psi_1}^N$ denote the disjunct corresponding to N in the formula above. By induction, we have the following equivalence: the interpretation map $(Z_{z+1}^j)^{j\leqslant m} \mapsto (U_{z+1}^j)^{j\leqslant m}$ leads to satisfaction of $\varphi_{\psi_1}^N$ in the $\tilde{\mathcal{Z}}$-extended variant of G if and only if

$$Z_{z+1} \mapsto \bigcup_{1\leqslant j\leqslant m}\big(\{j\}\times U_{z+1}^j\big) \cup N$$

is a satisfying choice for ψ_1 in the $\mathcal{Z}$-extended variant of $\mu(G)$. ■

6.5.2 Getting rid of multiple edge relations

We now show how to encode a multi-relational digraph into a 1-relational one, by inserting additional labeled nodes that represent the different edge relations.

Proposition 6.15 .
▶ For all $s, r \geqslant 0$ and $\Phi \in \{\overrightarrow{\mathrm{ML}}_g, \overleftrightarrow{\mathrm{ML}}_g\}$, there is a linear encoding μ from DG_s^r into DG_{s+r}^1 that allows for bidirectional translation within Φ.
Moreover, $\mu(\mathrm{DG}_s^r)$ is $\Pi_1^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}}_g)$-definable over DG_{s+r}^1. ◀

Proof. We choose μ to be the linear encoding that assigns to each s-bit labeled, r-relational digraph G the $(s+r)$-bit labeled (1-relational) digraph $\mu(G) = H$ with domain $[r+1] \times V^G$, labeling sets $P_i^H = \{1\} \times P_i^G$, for $1 \leqslant i \leqslant s$, and $P_{s+i}^H = \{i+1\} \times V^G$, for $1 \leqslant i \leqslant r$, and edge relation

$$\begin{aligned} R^H = \bigcup_{1\leqslant i\leqslant r}\Big(&\big\{\big((1,u),(i+1,u)\big) \;\big|\; u \in V^G\big\} \;\cup \\ &\big\{\big((i+1,u),(1,u)\big) \;\big|\; u \in V^G\big\} \;\cup \\ &\big\{\big((i+1,u),(i+1,u')\big) \;\big|\; (u,u') \in R_i^G\big\}\Big). \end{aligned}$$

That is, for each node $u \in V^G$ and for $1 \leqslant i \leqslant r$, we introduce an additional node representing the “R_i-port” of u, and connect everything accordingly.

Our forward translation, from Φ on DG^r_s to Φ on $\mu(\mathrm{DG}^r_s)$, is given by

$$\begin{aligned}
\psi_{P_i} &= P_i \quad \text{for } 1 \leqslant i \leqslant s,\\
\psi_{R_i} &= \Diamond\big(\psi_{i+1} \land \Diamond\Diamond(\psi_1 \land Y)\big) \quad \text{for } 1 \leqslant i \leqslant r,\\
\psi_{R_i^{-1}} &= \Diamond\big(\psi_{i+1} \land \overline{\Diamond}\Diamond(\psi_1 \land Y)\big) \quad \text{for } 1 \leqslant i \leqslant r,\\
\psi_{\bullet} &= \langle\!\cdot\!\rangle(\psi_1 \land Y),\\
\psi_{\mathrm{ini}} &= \psi_{\bullet},
\end{aligned}$$

$$\begin{aligned}
\text{where}\quad \psi_1 &= \neg \textstyle\bigvee_{1\leqslant i\leqslant r}(\psi_{i+1}) && \text{(“regular”)},\\
\psi_{i+1} &= P_{s+i} \quad \text{for } 1 \leqslant i \leqslant r && \text{(“}R_i\text{-port”)}.
\end{aligned}$$

Our translation in the other direction, from Φ on $\mu(\mathrm{DG}^r_s)$ to Φ on DG^r_s, is given by

$$\begin{aligned}
\varphi^{h+1}_{P_i} &= \begin{cases} P_i & \text{for } h = 0 \text{ and } 1 \leqslant i \leqslant s,\\ \bot & \text{for } h = 0 \text{ and } s+1 \leqslant i \leqslant s+r,\\ \top & \text{for } 1 \leqslant h \leqslant r \text{ and } i = s+h,\\ \bot & \text{for } 1 \leqslant h \leqslant r \text{ and } i \neq s+h,\end{cases}\\
\varphi^{h+1}_{R} &= \begin{cases} \bigvee_{1\leqslant i\leqslant r} X^{i+1} & \text{for } h = 0,\\ X^1 \lor \langle h\rangle X^{h+1} & \text{for } 1 \leqslant h \leqslant r,\end{cases}\\
\varphi^{h+1}_{R^{-1}} &= \begin{cases} \bigvee_{1\leqslant i\leqslant r} X^{i+1} & \text{for } h = 0,\\ X^1 \lor \overline{\langle h\rangle} X^{h+1} & \text{for } 1 \leqslant h \leqslant r,\end{cases}\\
\varphi^{h+1}_{\bullet} &= \langle\!\cdot\!\rangle(X^1 \lor \ldots \lor X^{r+1}) \quad \text{for } 0 \leqslant h \leqslant r,\\
\varphi_{\mathrm{ini}} &= \varphi^1_{\bullet}.
\end{aligned}$$

We can characterize $\mu(\mathrm{DG}^r_s)$ over the class DG^1_{s+r} by the conjunction of the following $\Pi^{\mathrm{MSO}}_1(\overrightarrow{\mathrm{ML}}_g)$-definable properties, using our helper formulas $(\psi_i)_{1\leqslant i\leqslant r+1}$ from the forward translation.

- A “port” that corresponds to a relation symbol R_i may not be associated with any other relation symbol R_j, nor be labeled with predicates $(P_j)_{1\leqslant j\leqslant s}$.

$$\bigwedge_{1\leqslant i\leqslant r} \boxdot\Big(\psi_{i+1} \to \neg\bigvee_{1\leqslant j\leqslant r,\, j\neq i}(\psi_{j+1}) \land \neg\bigvee_{1\leqslant j\leqslant s}(P_j)\Big)$$

- Every “regular node” is connected to its r different “ports”, and nothing else. The uniqueness of each “R_i-port” can be expressed by the $[\Pi^{\mathrm{MSO}}_1(\overrightarrow{\mathrm{ML}}_g)]$-formula $\mathrm{see1}(\psi_{i+1})$, using the construction from Example 2.2 in Section 2.6.

$$\boxdot\Big(\psi_1 \to \neg\Diamond\psi_1 \land \bigwedge_{1\leqslant i\leqslant r} \mathrm{see1}(\psi_{i+1})\Big)$$

- Similarly, each “port” is connected to precisely one “regular node” and to an arbitrary number of “ports” of the same relation symbol, but not to any other ones.

$$\bigwedge_{1\leqslant i\leqslant r} \boxdot\Big(\psi_{i+1} \to \mathrm{see1}(\psi_1) \land \neg\bigvee_{1\leqslant j\leqslant r,\, j\neq i}\Diamond\psi_{j+1}\Big)$$

- Finally, the links between "regular nodes" and "ports" have to be bidirectional: for each edge from a node of one type to a node of a different type, the corresponding inverse edge must also exist.

$$\bigwedge_{1\leqslant i\leqslant r+1} \forall X \boxdot\big(\psi_i \wedge X \rightarrow \Box(\neg\psi_i \rightarrow \Diamond X)\big)$$

Note that, in combination with the previous properties, this ensures that we have the same total number of nodes for each type $i \in [1:r+1]$. ■

6.5.3 Getting rid of vertex labels

Being able to eliminate multiple edge relations at the cost of additional labeling sets (see Proposition 6.15), our natural next step is to encode labeled digraphs into unlabeled ones.

Proposition 6.16 .
▶ For all $s \geqslant 1$ and $\Phi \in \{\overrightarrow{\text{ML}}_g, \overleftrightarrow{\text{ML}}_g\}$, there is a linear encoding μ from DG_s^1 into DG that allows for bidirectional translation within Φ.

Moreover, $\mu(\text{DG}_s^1)$ is $\Pi_1^{\text{MSO}}(\overrightarrow{\text{ML}}_g)$-definable over DG. ◀

Proof. We construct the linear encoding μ that assigns to each s-bit labeled digraph G the (unlabeled) digraph $\mu(G) = H$ with domain $(\{1\} \times V^G) \cup [2:s+3]$ and edge relation

$$\begin{aligned} R^H = \ & \big\{\big((1,u),(1,u')\big) \,\big|\, (u,u') \in R^G\big\} \\ \cup\ & \big\{\big((1,u),3\big) \,\big|\, u \in V^G\big\} \\ \cup\ & \bigcup_{1\leqslant i\leqslant s}\big\{\big((1,u),\, i+3\big) \,\big|\, u \in P_i^G\big\} \\ \cup\ & \big\{(i+3,\, i+2) \,\big|\, 1 \leqslant i \leqslant s\big\} \\ \cup\ & \big\{(i+3,\, 2) \,\big|\, 0 \leqslant i \leqslant s\big\}. \end{aligned}$$

The idea is to introduce a gadget that contains a separate node for each labeling set of the original digraph, and then connect the "regular nodes" to this gadget in a way that corresponds to their respective labeling. The gadget is easily identifiable because it contains the only node in the digraph that has no outgoing edge (namely, node 2). We ensure this by connecting all the "regular nodes" to node 3.

Our forward translation, from Φ on DG_s^1 to Φ on $\mu(\text{DG}_s^1)$, is given by

$$\begin{aligned} \psi_{P_i} &= \Diamond\psi_{i+3} \quad \text{for } 1 \leqslant i \leqslant s, \\ \psi_R &= \Diamond(\psi_1 \wedge Y), \\ \psi_{R^{-1}} &= \overline{\Diamond}Y, \\ \psi_\bullet &= \diamonddot(\psi_1 \wedge Y), \\ \psi_{\text{ini}} &= \psi_\bullet, \end{aligned}$$

$$\begin{aligned} \text{where} \quad \psi_1 &= \neg \textstyle\bigvee_{2\leqslant i\leqslant s+3}(\psi_i), \\ \psi_2 &= \Box\bot, \\ \psi_3 &= \Diamond\psi_2 \wedge \Box\psi_2, \\ \psi_{i+3} &= \Diamond\psi_2 \wedge \Diamond\psi_{i+2} \wedge \Box(\psi_2 \vee \psi_{i+2}) \quad \text{for } 1 \leqslant i \leqslant s. \end{aligned}$$

Our translation in the other direction, from Φ on $\mu(\mathrm{DG}_s^1)$ to Φ on DG_s^1, is given by

$$\varphi_R^h = \begin{cases} \Diamond X^1 \vee X^3 \vee \\ \quad \bigvee_{1 \leqslant i \leqslant s} (P_i \wedge X^{i+3}) & \text{for } h = 1, \\ \bot & \text{for } h = 2, \\ X^2 & \text{for } h = 3, \\ X^2 \vee X^{h-1} & \text{for } 4 \leqslant h \leqslant s + 3, \end{cases}$$

$$\varphi_{R^{-1}}^h = \begin{cases} \bar{\Diamond} X^1 & \text{for } h = 1, \\ \bigvee_{0 \leqslant i \leqslant s} X^{i+3} & \text{for } h = 2, \\ \langle\bullet\rangle X^1 \vee X^{h+1} & \text{for } h = 3, \\ \langle\bullet\rangle (P_{h-3} \wedge X^1) \vee X^{h+1} & \text{for } 4 \leqslant h \leqslant s + 2, \\ \langle\bullet\rangle (P_{h-3} \wedge X^1) & \text{for } h = s + 3, \end{cases}$$

$$\varphi_\bullet^h = \langle\bullet\rangle (X^1 \vee \ldots \vee X^{s+3}) \quad \text{for } 1 \leqslant h \leqslant s + 3,$$

$$\varphi_{\mathrm{ini}} = \varphi_\bullet^1.$$

Using the helper formulas $(\psi_i)_{1 \leqslant i \leqslant s+3}$ from the forward translation, we can characterize $\mu(\mathrm{DG}_s^1)$ over DG as

$$\langle\bullet\rangle \psi_1 \wedge \bigwedge_{2 \leqslant i \leqslant s+3} \mathrm{tot1}(\psi_i) \wedge \boxdot(\psi_1 \to \Diamond\psi_3 \wedge \neg\Diamond\psi_2).$$

Here, each $[\Pi_1^{\mathrm{MSO}}(\vec{\mathrm{ML}}_g)]$-subformula $\mathrm{tot1}(\psi_i)$ is obtained through the singleton construction from Example 2.2 in Section 2.6. ∎

6.5.4 Getting rid of backward modalities

For the sake of completeness, we also describe the encoding that lets us simulate backward modalities by means of an additional edge relation.

Proposition 6.17 .
▶ There is a linear encoding μ from DG into DG_0^2 that allows for bidirectional translation between $\overleftrightarrow{\mathrm{ML}}_g$ and $\vec{\mathrm{ML}}_g$.
Moreover, $\mu(\mathrm{DG})$ is $\Pi_1^{\mathrm{MSO}}(\vec{\mathrm{ML}}_g)$-definable over DG_0^2. ◀

Proof. The encoding is straightforward: to each digraph G, we assign a copy $\mu(G) = H$ that is enriched with a second edge relation, which coincides with the inverse of the first. Formally, $V^H = \{1\} \times V^G$,

$$R_1^H = \big\{\big((1,u),(1,u')\big) \;\big|\; (u,u') \in R^G\big\}, \quad \text{and}$$
$$R_2^H = \big\{(v',v) \;\big|\; (v,v') \in R_1^H\big\}.$$

With this, in order to translate between $\overleftrightarrow{\mathrm{ML}}_g$ on DG and $\vec{\mathrm{ML}}_g$ on $\mu(\mathrm{DG})$, we merely have to replace backward modalities by R_2-modalities, and vice versa. Hence, when we fix our forward translation, we choose $\psi_R = \langle 1\rangle Y$ and $\psi_{R^{-1}} = \langle 2\rangle Y$, and for the backward translation we set $\varphi_{R_1}^1 = \Diamond X^1$ and $\varphi_{R_2}^1 = \bar{\Diamond} X^1$.
To define $\mu(\mathrm{DG})$ over DG_0^2, we can use the following $\Pi_1^{\mathrm{MSO}}(\vec{\mathrm{ML}}_g)$-formula:

$$\forall X \boxdot\big(X \to [1]\langle 2\rangle X \wedge [2]\langle 1\rangle X\big)$$ ∎

6.5.5 Getting rid of directed edges

In order to encode a digraph into an undirected graph, we proceed in a similar manner to the elimination of multiple edge relations in Proposition 6.15. Using an ad-hoc trick, we can do this by introducing only one additional labeling set.

Proposition 6.18 .
▶ There is a linear encoding μ from $\textsc{dg}$ into $\textsc{graph}^1_1$ that allows for bidirectional translation between $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$ and $\overrightarrow{\mathrm{ML}}_{\mathrm{g}}$.

Moreover, $\mu(\textsc{dg})$ is $\Pi_1^{\mathrm{MSO}}(\overrightarrow{\mathrm{ML}}_{\mathrm{g}})$-definable over $\textsc{graph}^1_1$. ◀

Proof. A suitable choice for μ is to take the function that assigns to every digraph G the 1-bit labeled undirected graph $\mu(G) = H$ with domain $([3] \times V^G) \cup [4:6]$, labeling set $P^H = [4:6]$, and edge relation $R^H = \{(v, v') \mid \{v, v'\} \in E^H\}$, where

$$
\begin{aligned}
E^H = \; & \{\{(1,u),(2,u)\} \mid u \in V^G\} \\
\cup \; & \{\{(1,u),(3,u)\} \mid u \in V^G\} \\
\cup \; & \{\{(2,u),(3,u')\} \mid (u,u') \in R^G\} \\
\cup \; & \{\{(2,u), 4\} \mid u \in V^G\} \\
\cup \; & \{\{(3,u), 5\} \mid u \in V^G\} \\
\cup \; & \{\{5, 6\}\}.
\end{aligned}
$$

The idea is that we connect each original node $u \in V^G$ to two new nodes, which represent the “outgoing port” and “incoming port” of d, and use undirected edges between “ports” to simulate directed edges between “regular nodes”. In order to distinguish the different types of nodes, we connect them in different ways to the additional P-labeled nodes.

Our forward translation, from $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on $\textsc{dg}$ to $\overrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on $\mu(\textsc{dg})$, is given by

$$
\begin{aligned}
\psi_R &= \Diamond\big(\psi_2 \wedge \Diamond\Diamond(\psi_1 \wedge Y)\big), \\
\psi_{R^{-1}} &= \Diamond\big(\psi_3 \wedge \Diamond\Diamond(\psi_1 \wedge Y)\big), \\
\psi_{\bullet} &= \langle\cdot\rangle(\psi_1 \wedge Y), \\
\psi_{\mathrm{ini}} &= \psi_{\bullet},
\end{aligned}
$$

$$
\begin{aligned}
\text{where} \quad \psi_1 &= \neg(\psi_2 \vee \ldots \vee \psi_6) && (\text{“regular”}), \\
\psi_2 &= \Diamond\psi_4 && (\text{“outgoing”}), \\
\psi_3 &= \neg P \wedge \Diamond\psi_5 && (\text{“incoming”}), \\
\psi_4 &= P \wedge \neg\Diamond P \wedge \Diamond\neg P, \\
\psi_5 &= P \wedge \Diamond P \wedge \Diamond\neg P, \\
\psi_6 &= P \wedge \Diamond P \wedge \neg\Diamond\neg P.
\end{aligned}
$$

Our backward translation, from $\overrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on $\mu(\textsc{dg})$ to $\overleftrightarrow{\mathrm{ML}}_{\mathrm{g}}$ on $\textsc{dg}$, is given by

$$
\varphi_P^h = \begin{cases} \bot & \text{for } 1 \leqslant h \leqslant 3, \\ \top & \text{for } 4 \leqslant h \leqslant 6, \end{cases}
$$

$$\varphi_R^h = \begin{cases} X^2 \vee X^3 & \text{for } h = 1, \\ X^1 \vee \Diamond X^3 \vee X^4 & \text{for } h = 2, \\ X^1 \vee \bar{\Diamond} X^2 \vee X^5 & \text{for } h = 3, \\ \odot X^2 & \text{for } h = 4, \\ \odot X^3 \vee X^6 & \text{for } h = 5, \\ X^5 & \text{for } h = 6, \end{cases}$$

$$\varphi_\bullet^h = \odot(X^1 \vee \ldots \vee X^6) \quad \text{for } 1 \leqslant h \leqslant 6,$$

$$\varphi_{\mathrm{ini}} = \varphi_\bullet^1.$$

We can define $\mu(\text{DG})$ over GRAPH_1^1 with the following $[\Pi_1^{\text{MSO}}(\vec{\text{ML}}_g)]$-formula. It makes use of our helper formulas $(\psi_i)_{1 \leqslant i \leqslant 6}$ from the forward translation and the constructions $\text{see1}(\psi_i)$ and $\text{tot1}(\psi_i)$ from Example 2.2 in Section 2.6.

$$\begin{aligned} &\bigwedge_{4 \leqslant i \leqslant 6} \text{tot1}(\psi_i) \;\wedge \\ &\boxdot\big(\psi_2 \to \text{see1}(\psi_1) \wedge \Box(\psi_1 \vee \psi_3 \vee \psi_4)\big) \;\wedge \\ &\boxdot\big(\psi_3 \to \text{see1}(\psi_1) \wedge \Box(\psi_1 \vee \psi_2 \vee \psi_5)\big) \;\wedge \\ &\boxdot\big(\psi_1 \to \text{see1}(\psi_2) \wedge \text{see1}(\psi_3) \wedge \Box(\psi_2 \vee \psi_3)\big) \end{aligned}$$

The first line states that the three P-labeled nodes are unique, which forces 5 and 6 to be connected. The remaining lines ensure that each "port" is connected to exactly one "regular node", and, conversely, that every "regular node" is linked to precisely one "outgoing port" and one "incoming port". As a consequence, the number of "regular nodes" must be the same as the number of "ports" of each type. Furthermore, the formula restricts the types of neighbors each node can have, while the usage of the helper formulas ψ_2 and ψ_3 makes sure that the required connections to the P-labeled nodes are established. Finally, the fact that ψ_1 characterizes the "regular nodes" as the "remaining ones" guarantees that there are no unaccounted-for nodes. ■

6.5.6 Getting rid of global modalities

Our last encoding function lets us simulate global modalities by inserting a new node that is bidirectionally connected to all the "regular nodes".

Proposition 6.19 .
▶ There is a linear encoding μ from DG into @DG that allows for bidirectional translation between $\vec{\text{ML}}_g$ and $\vec{\text{ML}}$.

Furthermore, μ can be easily adapted into a linear encoding μ' from DG into DG that satisfies the following figurative inclusions, for arbitrary $\ell \geqslant 2$:

$$[\![\Sigma_\ell^{\text{MSO}}(\vec{\text{ML}}_g)]\!]_{\text{DG}} \;\vec{\subseteq}_{\mu'}\; [\![\boxdot\, \Sigma_\ell^{\text{MSO}}(\vec{\text{ML}})]\!]_{\text{DG}}\,,$$

$$[\![\Pi_\ell^{\text{MSO}}(\vec{\text{ML}}_g)]\!]_{\text{DG}} \;\vec{\equiv}_{\mu'}\; [\![\boxdot\, \Pi_\ell^{\text{MSO}}(\vec{\text{ML}})]\!]_{\text{DG}}\,. \quad ◀$$

Proof. We choose μ to be the linear encoding that maps each digraph G to the pointed digraph $\mu(G) = H$ with domain $(\{1\} \times V^G) \cup [2:3]$, position $@^H = 2$, and edge relation

$$\begin{aligned} R^{H} = \; & \{((1,u),(1,u')) \mid (u,u') \in R^{G}\} \\ & \cup \{((1,u),2) \mid u \in V^{G}\} \\ & \cup \{(2,(1,u)) \mid u \in V^{G}\} \\ & \cup \{(2,3)\}. \end{aligned}$$

One can distinguish node 2 from the others because it is connected to 3, which is the only node without any outgoing edge.

Our forward translation, from $\vec{\mathrm{ML}}_{\mathrm{g}}$ on DG to $\vec{\mathrm{ML}}$ on $\mu(\mathrm{DG})$, is given by

$$\begin{aligned} \psi_{R} &= \Diamond(\psi_1 \wedge Y), \\ \psi_{\bullet} &= \Diamond(\psi_2 \wedge \Diamond(\psi_1 \wedge Y)), \\ \psi_{\mathrm{ini}} &= \Diamond(\psi_1 \wedge Y), \end{aligned}$$

$$\text{where} \quad \psi_1 = \Diamond\Diamond\Box\bot \quad \text{and} \quad \psi_2 = \Diamond\Box\bot.$$

Our backward translation, from $\vec{\mathrm{ML}}$ on $\mu(\mathrm{DG})$ to $\vec{\mathrm{ML}}_{\mathrm{g}}$ on DG, is given by

$$\varphi_{R}^{h} = \begin{cases} \Diamond X^{1} \vee X^{2} & \text{for } h = 1, \\ \langle\!\cdot\!\rangle X^{1} \vee X^{3} & \text{for } h = 2, \\ \bot & \text{for } h = 3, \end{cases}$$

$$\varphi_{\mathrm{ini}} = \langle\!\cdot\!\rangle X^{2}.$$

Turning to the second claim of the proposition, we obtain $\mu'(G)$ by simply removing the position marker from $\mu(G)$, i.e., for every digraph G, $\mu'(G)$ is such that $\mu'(G)[@ \mapsto 2] = \mu(G)$.

For the forward figurative inclusions, let $\Xi \in \{\Sigma_{\ell}^{\mathrm{MSO}}, \Pi_{\ell}^{\mathrm{MSO}}\}$, for some arbitrary $\ell \geqslant 0$. By applying Lemma 6.14 a on μ, we get that for every $\Xi(\vec{\mathrm{ML}}_{\mathrm{g}})$-sentence φ over $\{R\}$, there is a $\Xi(\vec{\mathrm{ML}})$-sentence ψ_{φ} over $\{@, R\}$ such that, for all $G \in \mathrm{DG}$,

$$\begin{aligned} G \models \varphi \quad & \text{iff} \quad \mu'(G)[@ \mapsto 2] \models \psi_{\varphi}, \\ & \text{iff} \quad \mu'(G) \models \boxdot(\psi_2 \to \psi_{\varphi}). \end{aligned}$$

Hence, $[\![\Xi(\vec{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{DG}} \mathrel{\vec{\sqsubseteq}}_{\mu'} [\![\boxdot\, \Xi(\vec{\mathrm{ML}})]\!]_{\mathrm{DG}}$.

For the backward figurative inclusion, we require that $\ell \geqslant 2$. Slightly adapting the proof of Lemma 6.14 b to discard the part where we make use of the formula φ_{ini} from Definition 6.13 e (incidentally allowing us to merge the two consecutive induction proofs), it is easy to show the following: Given $h \in [3]$ and any $\Pi_{\ell}^{\mathrm{MSO}}(\vec{\mathrm{ML}})$-sentence ψ over $\{@, R\}$, we can construct a $\Pi_{\ell}^{\mathrm{MSO}}(\vec{\mathrm{ML}}_{\mathrm{g}})$-sentence φ_{ψ}^{h} over $\{@, R\}$ such that, for all $G \in \mathrm{DG}$ and $u \in V^{G}$,

$$G[@ \mapsto u] \models \varphi_{\psi}^{h} \quad \text{iff} \quad \mu'(G)[@ \mapsto v] \models \psi,$$

$$\text{where } v \text{ is } (h,u) \text{ if } h = 1, \text{ and } h \text{ otherwise.}$$

This immediately gives us a way of translating $\boxdot\psi$:

$$G \models \boxdot(\varphi_{\psi}^{1} \wedge \varphi_{\psi}^{2} \wedge \varphi_{\psi}^{3}) \quad \text{iff} \quad \mu'(G) \models \boxdot\psi.$$

The left-hand side sentence can be transformed into prenex normal form by simulating the global box with a universal set quantifier. Checking that a given set is *not* a singleton can be done in $\Sigma_1^{\mathrm{MSO}}(\vec{\mathrm{ML}}_{\mathrm{g}})$, since the negation is $\Pi_1^{\mathrm{MSO}}(\vec{\mathrm{ML}}_{\mathrm{g}})$-expressible (see Example 2.2 in Section 2.6). Thus, the given formula is equivalent to a $\Pi_\ell^{\mathrm{MSO}}(\vec{\mathrm{ML}}_{\mathrm{g}})$-formula, and we obtain that $[\![\Pi_\ell^{\mathrm{MSO}}(\vec{\mathrm{ML}}_{\mathrm{g}})]\!]_{\mathrm{DG}} \mathrel{\vec{\sqsupseteq}_{\mu'}} [\![\boxdot\, \Pi_\ell^{\mathrm{MSO}}(\vec{\mathrm{ML}})]\!]_{\mathrm{DG}}$. ■

7 Perspectives

Coming to the end of this thesis, we discuss some ideas for future research. They can be separated into two categories: rather focused questions that directly follow up on the results presented here, and broader questions that aim at the bigger picture.

7.1 Focused questions

Let us start with the topics directly related to this work, following roughly the order of discussion in the document.

7.1.1 Is there an alternation level that covers first-order logic?

In Chapter 3, we have related the classes of digraph languages recognizable by our three flavors of LDA_g's to those definable in MSOL, EMSOL and FOL. As shown in Figure 3.6 on page 33, ALDA_g's cover FOL (as a direct consequence of their equivalence to MSOL), whereas NLDA_g's do not. It is also easy to see that every NLDA_g is equivalent to an NLDA_g of length 1, since each node can simply guess all of its nondeterministic transitions at once, and then verify in one round of communication that its own choices are consistent with those of its neighbors. Furthermore, we know from Chapter 6 (Theorem 6.2 on page 66) that ALDA_g's of length $\ell + 1$ are strictly more expressive than ALDA_g's of length ℓ (or equivalently, that the set quantifier alternation hierarchy of $\text{MSO}(\overleftarrow{\text{ML}}_\text{g})$ is strict over digraphs). This means that length-restricted ALDA_g's form an infinite hierarchy of automata classes between NLDA_g's and ALDA_g's.

Against this backdrop, a natural question is whether there exists a bound ℓ such that ALDA_g's of length ℓ can recognize all languages definable in FOL on arbitrary digraphs. Note that this would also imply that ALDA_g's of length $\ell + 1$ fully cover EMSOL.

When restricted to digraphs of bounded degree, the answer is positive. This can be seen using Hanf's locality theorem, which basically states that on digraphs of bounded degree, every FOL-formula is equivalent to a Boolean combination of conditions of the form "r-sphere H occurs at least n times", where an r-sphere is a pointed digraph that represents the r-neighborhood of its distinguished node (see, e.g., [Tho96, Thm 4.1] or [Lib04, Thm 4.24]). Based on this characterization, it is relatively easy to show that any FOL-formula can be translated to an ALDA_{g} of length 3. However, for digraphs of unbounded degree, the author does not know the answer to the above question.

7.1.2 Does asynchrony entail quasi-acyclicity?

As already mentioned in Section 4.2, we make crucial use of quasi-acyclicity to prove the equivalence of a-QDA's and the backward μ-fragment (i.e., Theorem 4.2 on page 41). It is however open whether we really need to impose this condition on our asynchronous automata in order to be able to convert them into formulas of $\Sigma_1^{\mu}(\overleftarrow{\text{ML}})$. Asynchrony is a very strong requirement, and it might well be the case that every asynchronous automaton is in fact equivalent to a quasi-acyclic one. Moreover, if this assumption turned out to be true, it would be interesting to know if it extends to lossless-asynchronous automata.

7.1.3 Is asynchrony decidable?

Another natural question concerning asynchrony is whether there exists an algorithm that decides if a given distributed automaton is asynchronous, or alternatively, if it is lossless-asynchronous. Even though we can effectively translate from quasi-acyclic (lossless-)asynchronous automata to the backward μ-fragment (see Proposition 4.6), our translation procedure relies on the guarantee that the given automaton is indeed an la-QDA. From a practical perspective, it would be advantageous if the procedure could also check that its input is valid. While quasi-acyclicity can be easily verified, (lossless-)asynchrony seems to present a more challenging problem.

7.1.4 Are forgetful automata useful as tree automata?

In Chapter 5, we have seen that forgetful distributed automata are strictly more expressive than classical tree automata on ordered ditrees (Proposition 5.3 on page 52). Moreover, their emptiness problem is decidable on arbitrary digraphs (Theorem 5.4 on page 53), and since all distributed automata satisfy a tree-model property (see Lemma 5.6 on page 56), it is straightforward to adapt our decision procedure to the special case of ordered ditrees.

This begs the question whether forgetful automata could be of use in typical application areas of tree automata, such as program verification and processing of XML-like data. A first step towards an answer would be to investigate their closure properties (which are probably not as nice as those of tree automata) and to precisely analyze the complexities of their decision problems. Indeed, the LOGSPACE complexity of the emptiness problem (stated in Theorem 5.4) has to be revised for the case of ordered ditrees because distributed automata that operate exclusively on such structures do not have to deal with sets of states and thus can be represented more compactly; this leads to higher computational complexity.

7.1.5 How powerful are quasi-acyclic automata on dipaths?

We have presented two different constructions to prove the undecidability of the emptiness problem for distributed automata. The first (Theorem 5.5 on page 54) uses the idea of exchanging space and time to simulate a Turing machine by a distributed automaton that runs on a dipath. This simulation shows that even the dipath-emptiness problem is undecidable, but it works only if the state diagram of the simulating automaton may contain cycles. Our second approach (Theorem 5.9 on page 57) shows that also for quasi-acyclic automata the emptiness problem is undecidable, but the construction is much more technical and does not work if we restrict ourselves to dipaths.

It is part of ongoing work to establish a precise characterization of quasi-acyclic automata on dipaths in terms of counter machines. As a corollary, this will yield a stronger undecidability result that supersedes the two previous ones.

7.2 Broader questions

To conclude, let us expand our focus by suggesting possible extensions and asking how the present work fits into the wider landscape of graph automata and distributed computing.

7.2.1 What about distributed automata on infinite digraphs?

Although in this thesis we have considered only finite structures, this restriction is by no means necessary; distributed automata could also run on infinite digraphs, and this would not even require changing their definition. It is straightforward to see that the equivalence of ALDA_g's and MSOL established in Chapter 3 immediately extends to the infinite setting (simply by verifying that the given proofs remain applicable). However, this is not the case for all the results presented here. In particular in Chapter 4, we have relied on the fact that our digraphs are finite to prove the equivalence of quasi-acyclic asynchronous automata and the backward μ-fragment (see the proof of Proposition 4.3 on page 42). It seems that a more powerful acceptance condition would be required in order to get a corresponding equivalence on infinite digraphs.

For future research on distributed automata, it would be worthwhile to systematically consider both the finite and the infinite case.

7.2.2 What is the overlap with cellular automata?

Obviously, distributed automata are closely related to cellular automata. The only noteworthy difference is that distributed automata can operate on arbitrary digraphs, whereas cellular automata are usually confined to regular structures, such as (doubly linked) grids or dipaths. That is, if we restrict ourselves to the appropriate classes of digraphs, then the two models are exactly the same. Furthermore, there is a branch of research concerned with cellular automata as language acceptors (see, e.g., [Kut08, § 6.5] or [Ter12]). In order not to "reinvent the wheel", it is thus important to relate questions arising in the study of distributed automata to the existing body of knowledge in cellular automata theory.

An example where this was not done thoroughly enough can be found in Section 5.4. As the author has been recently informed by N. Bacquey (on 20 October 2017), the idea of exchanging space and time is well-known within the community of cellular automata. It is for instance documented in [Ter12, Fig. 9], where it is employed to simulate a real-time one-dimensional two-way cellular automaton by a corresponding one-way automaton. Although the presentation and purpose differ considerably from those in the present work, the technical construction is essentially the same.

7.2.3 Can we characterize more powerful models?

As mentioned at the beginning of Chapter 1, the original motivation for this thesis was to work toward a descriptive complexity theory for distributed computing. By focusing on distributed automata, we were able to make some progress in that direction, but the main challenge remains to establish logical characterizations of stronger models of computation, powerful enough to cover the kinds of algorithms usually considered in distributed computing. In order to be of practical interest, such a characterization should be in terms of *finite* formulas, just like the one provided by Fagin's theorem for nondeterministic polynomial-time Turing machines.

There are several ideas "in the air" on how one might characterize distributed finite-state machines equipped with unique identifiers, or even distributed Turing machines subject to certain time and space constraints. As of the time of writing, the author is not aware of any fully developed solution, but new results should be expected in the next few years.

Bibliography

[AM09] Rajeev Alur & P. Madhusudan (2009): *Adding nesting structure to words. J. ACM* 56(3), pp. 16:1–16:43, doi:10.1145/1516512.1516518.

[BB07] Patrick Blackburn & Johan van Benthem (2007): *Modal logic: a semantic perspective.* In Patrick Blackburn, Johan van Benthem & Frank Wolter, editors: *Handbook of Modal Logic*, *Studies in Logic and Practical Reasoning* 3, Elsevier, pp. 1–84, doi:10.1016/S1570-2464(07)80004-8.

[BDFO17] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud & Dennis Olivetti (2017): *What Can Be Verified Locally?* In Heribert Vollmer & Brigitte Vallée, editors: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, *LIPIcs* 66, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 8:1–8:13, doi:10.4230/LIPIcs.STACS.2017.8.

[Ben83] Johan van Benthem (1983): *Modal Logic and Classical Logic.* Indices: Monographs in Philosophical Logic and Formal Linguistics, Vol 3, Bibliopolis.

[BRV02] Patrick Blackburn, Maarten de Rijke & Yde Venema (2002): *Modal logic. Cambridge Tracts in Theoretical Computer Science* 53, Cambridge University Press, Cambridge, doi:10.1017/CBO9781107050884.

[BS07] Julian Bradfield & Colin Stirling (2007): *Modal mu-calculi.* In Patrick Blackburn, Johan van Benthem & Frank Wolter, editors: *Handbook of Modal Logic*, *Studies in Logic and Practical Reasoning* 3, Elsevier, pp. 721–756, doi:10.1016/S1570-2464(07)80015-2.

[Büc60] J. Richard Büchi (1960): *Weak Second-Order Arithmetic and Finite Automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6, pp. 66–92, doi:10.1002/malq.19600060105.

[Cat06] Balder ten Cate (2006): *Expressivity of Second Order Propositional Modal Logic. J. Philosophical Logic* 35(2), pp. 209–223, doi:10.1007/s10992-005-9012-9.

[CDG+08] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Christof Löding, Denis Lugiez, Sophie Tison & Marc Tommasi (2008):

Tree Automata Techniques and Applications. Available at http://tata.gforge.inria.fr. Release November 18, 2008.

[CE12] Bruno Courcelle & Joost Engelfriet (2012): *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach. Encyclopedia of mathematics and its applications* 138, Cambridge University Press, doi:10.1017/CBO9780511977619. Available at https://hal.archives-ouvertes.fr/hal-00646514.

[Cho56] Noam Chomsky (1956): *Three models for the description of language. IRE Trans. Information Theory* 2(3), pp. 113–124, doi:10.1109/TIT.1956.1056813.

[CKS81] Ashok K. Chandra, Dexter Kozen & Larry J. Stockmeyer (1981): *Alternation. J. ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.

[Cou90] Bruno Courcelle (1990): *The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. Inf. Comput.* 85(1), pp. 12–75, doi:10.1016/0890-5401(90)90043-H.

[DM97] Volker Diekert & Yves Métivier (1997): *Partial Commutation and Traces.* In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages: Volume 3 Beyond Words*, Springer Berlin Heidelberg, pp. 457–533, doi:10.1007/978-3-642-59126-6_8.

[Don70] John Doner (1970): *Tree Acceptors and Some of Their Applications. J. Comput. Syst. Sci.* 4(5), pp. 406–451, doi:10.1016/S0022-0000(70)80041-1.

[Elg61] Calvin C. Elgot (1961): *Decision Problems of Finite Automata Design and Related Arithmetics. Transactions of the American Mathematical Society* 98(1), pp. 21–51, doi:10.2307/1993511.

[Fag74] Ronald Fagin (1974): *Generalized First-Order Spectra and Polynomial-Time Recognizable Sets.* In Richard M. Karp, editor: *Complexity of Computation, SIAM-AMS Proceedings* 7, pp. 43–73. Available at http://www.almaden.ibm.com/cs/people/fagin/genspec.pdf.

[Fag75] Ronald Fagin (1975): *Monadic generalized spectra. Math. Log. Q.* 21(1), pp. 89–96, doi:10.1002/malq.19750210112.

[FF16] Laurent Feuilloley & Pierre Fraigniaud (2016): *Survey of Distributed Decision. Bulletin of the EATCS* 119. Available at http://bulletin.eatcs.org/index.php/beatcs/article/view/411/391.

[FFH16] Laurent Feuilloley, Pierre Fraigniaud & Juho Hirvonen (2016): *A Hierarchy of Local Decision.* In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani & Davide Sangiorgi, editors: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy, LIPIcs* 55, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 118:1–118:15, doi:10.4230/LIPIcs.ICALP.2016.118.

[GKL+07] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema & Scott Weinstein (2007): *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/3-540-68804-8.

[GR92] Dora Giammarresi & Antonio Restivo (1992): *Recognizable Picture Languages. IJPRAI* 6(2&3), pp. 241–256, doi:10.1142/S021800149200014X.

[GRST96] Dora Giammarresi, Antonio Restivo, Sebastian Seibert & Wolfgang Thomas (1996): *Monadic Second-Order Logic Over Rectangular Pictures and Recognizability by Tiling Systems. Inf. Comput.* 125(1), pp. 32–45, doi:10.1006/inco.1996.0018.

[GS11] Mika Göös & Jukka Suomela (2011): *Locally checkable proofs.* In Cyril Gavoille & Pierre Fraigniaud, editors: *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, ACM, pp. 159–168, doi:10.1145/1993806.1993829.

[GS16] Mika Göös & Jukka Suomela (2016): *Locally Checkable Proofs in Distributed Computing. Theory of Computing* 12(1), pp. 1–33, doi:10.4086/toc.2016.v012a019.

[HJK+12] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela & Jonni Virtema (2012): *Weak models of distributed computing, with connections to modal logic.* In Darek Kowalski & Alessandro Panconesi, editors: *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, ACM, pp. 185–194, doi:10.1145/2332432.2332466.

[HJK+15] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela & Jonni Virtema (2015): *Weak models of distributed computing, with connections to modal logic. Distributed Computing* 28(1), pp. 31–53, doi:10.1007/s00446-013-0202-3. Available at https://arxiv.org/abs/1205.2051.

[Imm99] Neil Immerman (1999): *Descriptive complexity.* Graduate texts in computer science, Springer, doi:10.1007/978-1-4612-0539-5.

[KK07] Amos Korman & Shay Kutten (2007): *Distributed verification of minimum spanning trees. Distributed Computing* 20(4), pp. 253–266, doi:10.1007/s00446-007-0025-1.

[Kle56] Stephen C. Kleene (1956): *Representation of events in nerve nets and finite automata.* In Claude Shannon & John McCarthy, editors: *Automata studies*, Annals of mathematics studies, no. 34, Princeton University Press, Princeton, N. J., pp. 3–41.

[KR17] Antti Kuusisto & Fabian Reiter (2017): *Emptiness Problems for Distributed Automata.* In Patricia Bouyer, Andrea Orlandini & Pierluigi San Pietro, editors: *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017., EPTCS* 256, pp. 210–222, doi:10.4204/EPTCS.256.15.

[Kut08] Martin Kutrib (2008): *Cellular Automata - A Computational Point of View.* In Gemma Bel Enguix, Maria Dolores Jiménez-López & Carlos Martín-Vide, editors: *New Developments in Formal Languages and Applications, Studies in Computational Intelligence* 113, Springer, pp. 183–227, doi:10.1007/978-3-540-78291-9_6.

[Kuu08] Antti Kuusisto (2008): *A modal perspective on monadic second-order alternation hierarchies.* In Carlos Areces & Robert Goldblatt, editors: *Advances in Modal Logic 7, papers from the seventh conference on "Advances in Modal Logic," 9-12 September 2008, Nancy, France*, College Publications, pp. 231–247. Available at http://www.aiml.net/volumes/volume7/Kuusisto.pdf.

[Kuu13a] Antti Kuusisto (2013): *Modal Logic and Distributed Message Passing Automata.* In Simona Ronchi Della Rocca, editor: *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, *LIPIcs* 23, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 452–468, doi:10.4230/LIPIcs.CSL.2013.452.

[Kuu13b] Antti Kuusisto (2013): *Modal Logics and Definability. TamPub, Institutional repository of the University of Tampere*, pp. 1–6. Available at http://urn.fi/URN:ISBN:978-951-44-9157-3.

[Kuu14] Antti Kuusisto (2014): *Infinite Networks, Halting and Local Algorithms.* In Adriano Peron & Carla Piazza, editors: *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, *EPTCS* 161, pp. 147–160, doi:10.4204/EPTCS.161.14.

[Kuu15] Antti Kuusisto (2015): *Second-order propositional modal logic and monadic alternation hierarchies. Ann. Pure Appl. Logic* 166(1), pp. 1–28, doi:10.1016/j.apal.2014.08.003.

[Len05] Giacomo Lenzi (2005): *The Modal μ-Calculus: a Survey. TASK Quarterly – Scientific Bulletin of the Academic Computer Centre in Gdansk* 9(3), pp. 293–316. Available at http://task.gda.pl/quart/05-3.html.

[Lib04] Leonid Libkin (2004): *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-662-07003-1.

[Löd12] Christof Löding (2012): *Basics on Tree Automata.* In Deepak D'Souza & Priti Shankar, editors: *Modern Applications of Automata Theory*, *IISc Research Monographs Series* 2, World Scientific, pp. 79–109, doi:10.1142/9789814271059_0003.

[LT00] Christof Löding & Wolfgang Thomas (2000): *Alternating Automata and Logics over Infinite Words.* In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses & Takayasu Ito, editors: *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, *Lecture Notes in Computer Science* 1872, Springer, pp. 521–535, doi:10.1007/3-540-44929-9_36.

[Lyn96] Nancy A. Lynch (1996): *Distributed Algorithms.* Morgan Kaufmann.

[Mat02] Oliver Matz (2002): *Dot-depth, monadic quantifier alternation, and first-order closure over grids and pictures. Theor. Comput. Sci.* 270(1-2), pp. 1–70, doi:10.1016/S0304-3975(01)00277-8.

[MST02] Oliver Matz, Nicole Schweikardt & Wolfgang Thomas (2002): *The Monadic Quantifier Alternation Hierarchy over Grids and Graphs. Inf. Comput.* 179(2), pp. 356–383, doi:10.1006/inco.2002.2955.

[MT97] Oliver Matz & Wolfgang Thomas (1997): *The Monadic Quantifier Alternation Hierarchy over Graphs is Infinite.* In: *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, IEEE Computer Society, pp. 236–244, doi:10.1109/LICS.1997.614951.

[Ner58] Anil Nerode (1958): *Linear automaton transformations. Proc. Amer. Math. Soc.* 9, pp. 541–544, doi:10.2307/2033204.

[Pel00] David Peleg (2000): *Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications* 5, Society for Industrial and Applied Mathematics (SIAM), doi:10.1137/1.9780898719772.

[Rei15] Fabian Reiter (2015): *Distributed Graph Automata.* In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, IEEE Computer Society, pp. 192–201, doi:10.1109/LICS.2015.27. Available at https://arxiv.org/abs/1408.3030.

[Rei16] Fabian Reiter (2016): *Alternating Set Quantifiers in Modal Logic. CoRR* abs/1602.08971. Available at http://arxiv.org/abs/1602.08971.

[Rei17] Fabian Reiter (2017): *Asynchronous Distributed Automata: A Characterization of the Modal Mu-Fragment.* In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn & Anca Muscholl, editors: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, *LIPIcs* 80, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 100:1–100:14, doi:10.4230/LIPIcs.ICALP.2017.100. Available at http://arxiv.org/abs/1611.08554.

[SB99] Thomas Schwentick & Klaus Barthelmann (1999): *Local Normal Forms for First-Order Logic with Applications to Games and Automata. Discrete Mathematics & Theoretical Computer Science* 3(3), pp. 109–124. Available at http://dmtcs.episciences.org/254.

[Sch97] Nicole Schweikardt (1997): *The Monadic Quantifier Alternation Hierarchy over Grids and Pictures.* In Mogens Nielsen & Wolfgang Thomas, editors: *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, *Lecture Notes in Computer Science* 1414, Springer, pp. 441–460, doi:10.1007/BFb0028030.

[Suo13] Jukka Suomela (2013): *Survey of local algorithms. ACM Comput. Surv.* 45(2), pp. 24:1–24:40, doi:10.1145/2431211.2431223.

[Ter12] Véronique Terrier (2012): *Language Recognition by Cellular Automata.* In Grzegorz Rozenberg, Thomas Bäck & Joost N. Kok, editors: *Handbook of Natural Computing*, Springer, pp. 123–158, doi:10.1007/978-3-540-92910-9_4.

[Tho91] Wolfgang Thomas (1991): *On Logics, Tilings, and Automata.* In Javier Leach Albert, Burkhard Monien & Mario Rodríguez-Artalejo, editors: *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, *Lecture Notes in Computer Science* 510, Springer, pp. 441–454, doi:10.1007/3-540-54233-7_154.

[Tho96] Wolfgang Thomas (1996): *Elements of an automata theory over partial orders.* In Doron A. Peled, Vaughan R. Pratt & Gerard J. Holzmann, editors: *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, July 24-26, 1996*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 29, DIMACS/AMS, pp. 25–40.

[Tho97a] Wolfgang Thomas (1997): *Automata Theory on Trees and Partial Orders.* In Michel Bidoit & Max Dauchet, editors: *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France, April 14-18, 1997, Proceedings*, *Lecture Notes in Computer Science* 1214, Springer, pp. 20–38, doi:10.1007/BFb0030586.

[Tho97b] Wolfgang Thomas (1997): *Languages, Automata, and Logic.* In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages: Volume 3 Beyond Words*, Springer, Berlin, pp. 389–455, doi:10.1007/978-3-642-59126-6_7.

[Tra61] Boris A. Trakhtenbrot (1961): *Finite automata and the logic of single-place predicates.* *Soviet Physics Dokl.* 6, pp. 753–755.

[TW68] James W. Thatcher & Jesse B. Wright (1968): *Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic.* *Mathematical Systems Theory* 2(1), pp. 57–81, doi:10.1007/BF01691346.

Index

$\vec{\equiv}_{\mu}$, 68–71, 88
$\vec{\sqsubseteq}_{\mu}$, 68, 69, 71, 79, 88, 89
$\vec{\sqsupseteq}_{\mu}$, 68, 70, 79, 90
f^{-1}, 68–70
R^{-1}, 16, 76, 78, 80, 82, 84–87
$\rightrightarrows$, 44, 45
$\triangleright$, 43–48
$|S|$, 11, 14, 25, 28, 32, 34, 46, 54, 56, 58, 59, 73
$|x|$, 11, 46, 59
$]m:n]$, 11, 77, 83
$[m:n]$, 11–14, 25, 31, 77, 78, 80, 81, 83, 85, 87–89
$[\Phi]_{\mathcal{C}}$, 17, 19, 71, 75, 84, 86, 88
$[\varphi]_{\mathcal{C}}$, 17
$[\![\mathcal{A}]\!]_{\mathcal{C}}$, 19, 20, 41, 42, 46, 51, 52
$[\![\mathcal{A}]\!]_{\mathcal{C}}$, 25, 28–34
$[\![A]\!]_{\mathcal{C}}$, 19
$[\![A]\!]_{\mathcal{C}}$, 25, 26, 28, 31, 32
$[\![\varphi]\!]_{G}$, 17, 39, 40
$[\![\Phi]\!]_{\mathcal{C}}$, 9, 17, 20, 30, 31, 33, 34, 40–42, 46, 51, 52, 65–73, 75, 79, 88–90
$[\![\varphi]\!]_{\mathcal{C}}$, 17, 25, 40
$[\![\mathcal{T}]\!]_{\mathcal{C}}$, 72, 73, 75
$\Delta_{\ell}^{\mathrm{MSO}}$, 65–67, 69, 70, 75
$\Pi_{\ell}^{\mathrm{MSO}}$, 19, 65–67, 69–71, 75, 79, 83–90
$\Sigma_{\ell}^{\mathrm{MSO}}$, 9, 15, 34, 63, 65–67, 69–73, 75, 79, 88–90
Σ_{ℓ}^{μ}, 39–42, 46, 92
$\varphi(S)$, 16, 30, 34
$\varphi[X \mapsto \psi]$, 72, 79–83
S^{G}, 12–19, 25, 34, 38, 39, 42–44, 46–48, 51, 76, 81, 83, 85–89
λ^{G}, 13, 14, 19, 25, 28, 30, 31, 38, 39, 42–44, 46–48, 51
V^{G}, 12–19, 24, 25, 28, 30–32, 38–40, 42–44, 46–48, 51, 73, 74, 76–83, 85–89
$G[\alpha]$, 12, 13, 31
$G[\zeta]$, 13, 24, 25, 28, 30–32
$G[\nu]$, 13, 14, 19, 38–40, 44, 46–48, 50, 51, 53, 56, 57
$G[S \mapsto I]$, 12, 13, 15, 17, 18, 39, 77–82, 89
$\doteq$, 15, 31, 72
x, 15–17, 72, 80, 81
X, 15–18, 39, 40, 42, 72–74, 76–82, 84–89
$X(x)$, 15, 18, 31, 72
$R(x_0, \ldots, x_k)$, 15, 18, 31, 72
$\exists_S$, 15, 17, 18, 30, 31, 65, 72, 74, 75, 81, 83
$\forall_S$, 16, 18, 30, 76, 85, 86
μ, 39, 40, 42, 46
$\overline{\boxed{R}}$, 16, 39, 40, 42, 46, 76
$\overline{\Diamond R}$, 15, 16, 39, 40, 42, 46, 72, 80, 82, 84–86, 88
$\boxed{R}$, 16, 18, 74, 76, 85, 86, 88, 89
$\Diamond R$, 15–18, 72, 74, 76–78, 80, 82, 84–89
$\boxdot$, 16, 65, 74, 76, 84–86, 88, 89
$\Diamond\!\cdot$, 15, 16, 72, 74, 76, 78–89
$\bot$, 16, 39, 74, 76, 83–87, 89
$\top$, 16, 39, 76, 83, 84, 87
$\models$, 15–17, 31, 40, 43, 71, 77–82, 89
$\not\models$, 16
$\neg$, 15–18, 31, 39, 46, 65, 72–74, 80, 82,

84–87
$\wedge$, 16, 18, 30, 39, 40, 46, 72–74, 76, 84–89
$\vee$, 15–18, 31, 39, 40, 72, 74, 80, 82–89
$\leftrightarrow$, 16, 76
$\rightarrow$, 16, 18, 30, 74, 76, 84–86, 88, 89
$\varnothing$, 11, 14, 23–25, 40, 42, 51–54, 78, 79, 81, 83
$\bullet$, 16, 18, 78, 80, 82, 84–89
$\mathbb{2}$, 11, 13, 38, 42, 46, 72, 73
$\mathbb{2}^S$, 11, 19, 20, 23–25, 28, 31, 39, 40, 42–44, 46, 51, 53, 54, 68, 69
@, 12, 13, 15–18, 71, 72, 77–83, 88, 89
@DG, 9, 13, 14, 19, 20, 40–42, 46, 66, 67, 71, 88
@DIPATH, 14, 51
@ODITREE, 14, 52
accept, 8, 19, 39, 49–53, 55–58, 60, 61, 93
accepting configuration, 24–26, 29, 30
accepting run, 25–29, 32
accepting set, 22–24, 26, 29
accepting state, 19, 38, 42, 51, 52, 55–57, 59, 60
a-DA, 39, 41
adjacent, 13, 14, 18, 28, 56
ALDA$_g$, iii, 9, 22–34, 49, 63, 91–93
a-QDA, 39, 41, 42, 49, 50, 92
arity, 12, 13, 77, 78, 80, 82
asynchronous automaton, iii, 7–9, 37, 39–42, 92, 93
asynchronous run, 38, 42, 44, 46, 47
backward μ-fragment, iii, 5, 7–9, 37, 39–43, 46, 92, 93
backward bisimilar, 44, 47, 48
backward bisimulation, 43, 44
backward diamond, 16
backward figurative inclusion, 68, 89
backward modal logic, 5, 17, 20
backward translation, 78, 79, 82, 86, 87, 89
BC, 9, 65–67, 69, 70, 75, 79
bidirectional modal logic, 17
bidirectional translation, 79, 83, 85–88
box, 16, 65, 67, 90
colorable, 14, 18, 22, 23, 34
coloring, 1, 14, 18, 22, 23, 26, 27, 34
configuration, 19, 51
connected, 3, 6, 14, 26, 28, 32, 34, 71, 76
consistent, 39
constant, 12, 16, 39, 42
DA, 19, 20, 39, 41
definable, 2, 6, 17, 20, 21, 30, 33, 34, 37, 41, 64, 70–73, 75, 83–87, 91
define, 1, 5, 17–19, 25, 40
DG, 13, 14, 17, 18, 25, 26, 28–34, 66, 67, 70, 71, 75, 83–90
diamond, 16
digraph, iii, 1–5, 7, 8, 11–14, 16, 18–32, 34, 35, 37–39, 41–51, 53, 54, 59, 60, 64, 66, 70, 71, 75, 76, 83, 85–89, 91–93
digraph language, 1, 3, 5, 6, 14, 18, 21–23, 25–34, 63, 64, 91
dipath, 14, 50, 51, 54–58, 92, 93
dipath-emptiness problem, 50, 54, 56, 93
distinguished node, 12–14, 51, 57, 58, 91
distributed automaton, iii, 5, 7–9, 19, 20, 37–44, 49–56, 58, 60, 92–94
ditree, 6, 14, 52, 53, 56, 58–61
DLDA$_g$, iii, 25, 29, 32–35
domain, 2, 12–14, 17, 77, 83, 85, 87, 88
edge, 13, 14, 18, 29–32, 34, 38, 40, 43–46, 48, 53, 73, 76, 85, 87, 89
edge relation, 3, 13, 14, 16, 19, 22, 26, 34, 52, 83, 85–87, 89
emptiness problem, iii, 8, 9, 49, 50, 53, 54, 56, 57, 92, 93
EMSOL, 3, 4, 6, 7, 15, 16, 18, 33, 34, 63, 91
enriched configuration, 44–46
enriched run, 44–48
enriched run segment, 45, 46
existential configuration, 24–26, 30
existential monadic second-order logic, 16
existential state, 22–24, 26, 28, 30
extended variant, 12, 13, 39, 81, 83
fairness property, 38, 43
FDA, 50–52
figurative inclusion, 68–71, 77, 79, 88, 89
figuro, 68

first, 38, 44–48
first-order logic, 2, 12, 16, 66, 71, 72
fixpoint, 39, 40, 43
FOL, 15–18, 30, 33, 34, 65, 66, 69, 71, 72, 75, 91, 92
forgetful automaton, iii, 8, 49–54, 92
formula, iii, 1–3, 5, 7, 8, 12, 14–20, 30, 31, 38–40, 42, 43, 46, 47, 53, 63–66, 68, 69, 72–75, 77–84, 86, 88–90, 92, 94
forward figurative equality, 68
forward figurative inclusion, 68, 89
forward translation, 77, 79, 80, 84–89
free, 14–16, 18, 31, 72
free_0, 31
free, 14, 16, 30, 31, 72, 79, 81, 82
global diamond, 16
global transition function, iii, 24, 25
GRAPH, 14, 66, 67, 70, 87, 88
graph, 6, 13, 32, 66, 70, 87
GRID, 14, 66, 67, 69, 70, 72, 73, 75
grid, 14, 64–66, 69–75, 93
id, 68–70
incoming neighbor, iii, 5, 13, 14, 17, 19, 22, 26, 27, 37, 42–45, 47–50, 53, 56–58, 91
incoming neighborhood, 5, 13, 22, 26, 48, 50, 56, 60
initial configuration, 25, 30
initial state, 51
initialization function, 19, 42
interpretation, 12, 16, 17, 30, 31, 40, 47, 74, 80, 82, 83
kernel, 65–67, 69, 70, 75, 79–81, 83
l_A, 24
labeling, 3, 13, 14, 19, 22–32, 34, 38, 40, 42, 43, 46, 50–54, 58–60, 66, 70–75, 83–85, 87, 88
labeling set, 13, 14, 83, 85, 87
la-DA, 39, 41
language, 3, 12, 14, 17, 19, 21, 34, 53
la-QDA, 39, 41, 46, 49, 50, 92
last, 38, 44–47
LDA, 20, 34, 49, 63
LDA_g, 21–23, 25, 26, 33, 91
least fixpoint, iii, 39, 40, 42, 46, 47
len, 24, 32
linear encoding, 77–79, 81, 83, 85–88
local automaton, iii, 7–9, 20, 21, 49, 50, 63
lossless-asynchronous automaton, 39, 41, 43, 46, 92
lossless-asynchronous timing, 38, 39, 44, 46–48
lossless-asynchronous timing segment, 44, 45, 48
$\overrightarrow{\text{ML}}$, 9, 15, 17, 19, 63–67, 71, 73, 77–79, 88–90
$\overleftarrow{\text{ML}}$, 15, 17, 20, 34, 39–42, 46, 49, 63, 92
$\overleftrightarrow{\text{ML}}$, 15, 17, 19, 65, 77–79
$\overrightarrow{\text{ML}}_g$, 15, 17–19, 64–67, 69–71, 73–75, 77–79, 83–90
$\overleftarrow{\text{ML}}_g$, 15, 17, 34, 63, 64, 91
$\overleftrightarrow{\text{ML}}_g$, 15, 17–19, 65–67, 69–72, 74, 75, 77–79, 83, 85–87
modal logic, iii, viii, 5, 8, 12, 14, 17, 18, 63, 64, 66, 73
modal logic with global modalities, 17
modality, 5, 15–19, 34, 39, 42, 64, 66, 67, 70, 73, 78, 80–82, 86, 88
monadic second-order logic, iii, 2, 3, 16
monotonic, 40, 43
monovisioned automaton, 50, 56, 57
MSO, 15, 17, 18, 34, 63–67, 69, 71, 74, 75, 79, 91
MSOL, 2, 3, 7–9, 15–18, 20–22, 25, 30–34, 37, 41, 49, 51, 52, 63–66, 69, 71–74, 91, 93
$\mathbb{N}$, 11, 13, 19, 24, 38–40, 42–44, 47, 48, 51–54, 56, 57
$\mathbb{N}_+$, 11, 14, 38, 45
neighbor, 4, 6–8, 13, 88
neighborhood, 7, 13, 26, 27, 32
NLDA_g, iii, 22, 25, 28, 29, 32–35, 49, 63, 91
node, iii, 3–8, 12–14, 16–32, 34, 37, 38, 40, 42–61, 73, 74, 76–78, 80–85, 87–89, 91
node quantifier, 16–18, 63–65
node symbol, 12, 16, 17, 31, 72, 77, 78, 80, 82
nonlocal automaton, iii, 5, 7–9, 19, 37, 49, 50
nonpermanent state, 22–24
outgoing neighbor, 13, 14, 17, 19, 25,

32
outgoing neighborhood, 13
permanent configuration, 24–26, 30
permanent state, 21–24, 26
pointed digraph, 13, 14, 19, 21, 38, 40, 44, 47, 48, 50, 51, 53, 56–58, 60, 63, 64, 66, 67, 71, 88, 91
pointed dipath, 2–4, 7, 8, 14, 22, 28, 29, 34, 35, 49–52, 54, 55, 57
pointed ditree, 14, 52, 56–58
pointed ordered ditree, 3, 4, 7, 8, 14, 34, 35, 49–53, 92
pointed structure, 12, 13, 16–18, 64, 65
pointed variant, 13
pointed-digraph language, 14, 19, 20, 37, 40, 41, 50, 53, 54, 56–58, 93
popfirst, 38, 43, 45
position symbol, 12, 13, 16, 17, 80, 82, 83
prenex normal form, 65, 66, 90
projection, 28, 29, 31
pushlast, 38, 43–46
QDA, 20, 39, 41
quasi-acyclic automaton, 8, 9, 20, 38–43, 46, 50, 57, 58, 92, 93
reachable configuration, 25, 30
recognizable, 19, 20, 41, 57
recognize, 19, 37, 49
rejecting configuration, 24, 28, 30
relabeled variant, 13, 24
relation symbol, 12, 13, 16, 76–78, 80, 82, 84
root, 14, 48, 52, 53, 56–60
run, 19, 51–58
$\mathbb{S}_0$, 12, 15–17, 31, 72
$\mathbb{S}_k$, 12, 15–18, 31, 39, 65, 72, 78, 79, 81, 82
$\mathbb{S}$, 12, 16
satisfy, 16, 18, 30, 42, 46, 47, 73, 80–83
SB, 4, 5, 20
see1, 18, 84, 88
sentence, 16, 73, 77–83, 89, 90
set quantifier, iii, 3, 6, 8, 9, 15–18, 31, 33, 34, 63–67, 69, 71, 74, 75, 79, 80, 82, 90, 91
set symbol, 12, 13, 16, 31, 73, 75, 77–80, 82
sig, 12, 16, 18
signature, 12, 13, 18, 73, 77–80
sink, 13, 20, 25, 34, 50, 56, 60, 76
source, 13, 25, 76
state, iii, 8, 19, 20, 38, 42–45, 47–53, 55–60, 63, 92, 93
structure, 2, 3, 12–14, 16–18, 35, 44, 48, 58, 64, 65, 68–71, 75, 77–83, 92, 93
successor configuration, 25, 30
symbol, 12–14, 16, 18, 77, 80, 82
synchronous automaton, 39, 41
synchronous run, 38, 39, 44, 46
synchronous timing, 38
tiling system, 64, 72, 73, 75
timing, 38, 39, 42, 44, 47, 48
tot1, 18, 76, 86, 88
trace, 38, 44–48
transition function, 19, 42, 43, 51, 54, 55
TS, 72, 73, 75
universal configuration, 24–26, 30
universal state, 22–26, 28, 30
variable, 12, 16, 18, 30, 31, 39, 40, 42, 43, 46, 47, 79
$\mathbb{Z}$, 11