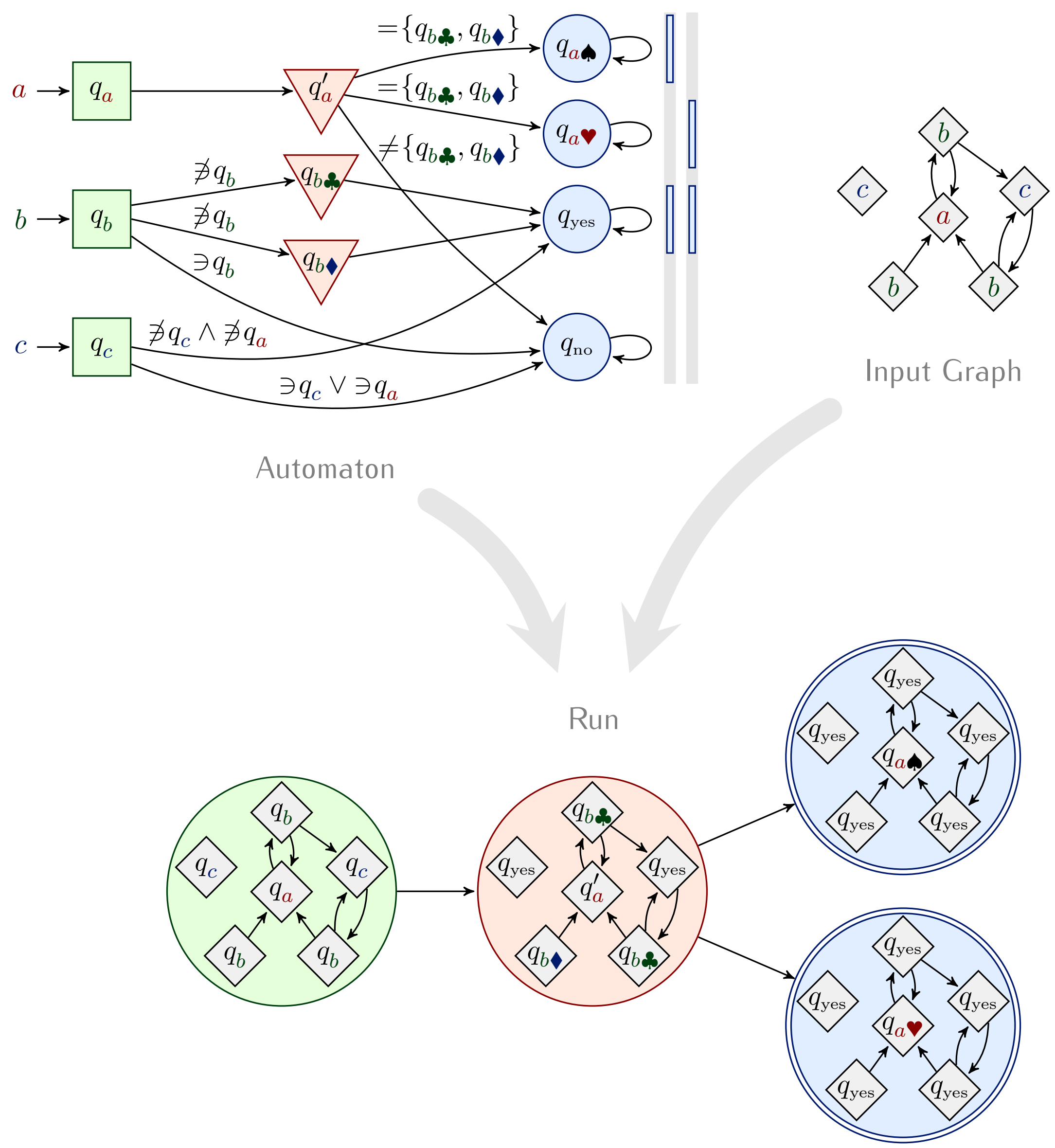
Fabian Reiter

Distributed Graph Automata



University of Freiburg

Master's Thesis in Computer Science

Supervisors

Prof. Dr. Fabian Kuhn (Chair of Algorithms and Complexity)
Prof. Dr. Andreas Podelski (Chair of Software Engineering)

Institution

Albert Ludwig University of Freiburg (Germany)
Faculty of Engineering
Department of Computer Science

Date of Submission (original version)

January 20th, 2014

Date of Revision (this version)

April 25th, 2014

Author and Contact Address

Fabian Reiter `fabian.reiter@gmail.com`

Abstract

Inspired by distributed algorithms, we introduce a new class of finite graph automata that recognize precisely the graph languages definable in monadic second-order logic. For the cases of words and trees, it has been long known that the regular languages are precisely those definable in monadic second-order logic. In this regard, the automata proposed in the present work can be seen, to some extent, as a generalization of finite automata to graphs.

Furthermore, we show that, unlike for finite automata on words and trees, the deterministic, nondeterministic and alternating variants of our automata form a strict hierarchy with respect to their expressive power. For the weaker variants, the emptiness problem is decidable.

Zusammenfassung

Inspiriert durch verteilte Algorithmen führen wir eine neue Klasse von endlichen Graph-Automaten ein, die genau die Graph-Sprachen erkennen, die in monadischer Prädikatenlogik zweiter Stufe definierbar sind. Für Worte und Bäume ist seit langem bekannt, dass die regulären Sprachen genau jene sind, die in monadischer Prädikatenlogik zweiter Stufe definierbar sind. In dieser Hinsicht können die in vorliegender Arbeit vorgestellten Automaten gewissermaßen als eine Verallgemeinerung von endlichen Automaten auf Graphen betrachtet werden.

Ferner zeigen wir, dass im Gegensatz zu endlichen Automaten auf Worten und Bäumen die deterministischen, nichtdeterministischen und alternierenden Varianten unserer Automaten eine strikte Hierarchie bezüglich ihrer Ausdrucksstärke bilden. Für die schwächeren Varianten ist das Leerheitsproblem entscheidbar.

Acknowledgments

I would like to thank my supervisors, Fabian Kuhn and Andreas Podelski, for many helpful and pleasant discussions, and their continuous support of a project whose outcome was unpredictable in its early stages. Furthermore, I am very grateful to Jan Leike, who kindly read and commented on drafts of this thesis.

Contents

Chapter 1

Introduction

The research for this thesis started with an open-ended (and perhaps naive) question: *what can we obtain by connecting finite automata in a synchronous distributed setting?* As it turns out, a possible answer is: a new class of automata that can be seen, to some extent, as a generalization of finite automata to graphs. In order to substantiate this claim, we begin by reviewing a fundamental result of formal language theory, and then use that result as a guide within the less well-explored world of graph languages.

1.1 Background and Related Work

In the early 1960s, a beautiful connection between automata theory and formal logic was discovered. Independently of each other, Büchi [Büc60], Elgot [Elg61] and Trakhtenbrot [Tra61] showed that the regular languages, recognized by finite automata, are precisely the languages defined by a certain class of logical formulas. This idea might be best understood through a simple example. The following one is borrowed from Thomas [Tho91].

1.1.1 Example.
Consider the nondeterministic finite automaton $\mathcal{A}_{bb}^{\mathrm{no}}$ specified in Fig. 1.1. If we exclude the empty word, this automaton accepts a finite word w over the alphabet $\Sigma = \{a, b\}$ if and only if w does not contain the segment bb and the last symbol of w is an a. We can define the same language by the following first-order formula:

$$\varphi_{bb}^{\mathrm{no}} := \neg\exists \mathsf{u}, \mathsf{v}\Big(\langle b\rangle \mathsf{u} \;\wedge\; \mathsf{u} \rightharpoonup \mathsf{v} \;\wedge\; \langle b\rangle \mathsf{v}\Big) \;\wedge\; \exists \mathsf{u}\Big(\langle a\rangle \mathsf{u} \;\wedge\; \neg\exists \mathsf{v}\big(\mathsf{u} \rightharpoonup \mathsf{v}\big)\Big)$$

The idea is that we identify each word with a labeled directed graph that consists of a single path. For instance, aba corresponds to $\langle a\rangle \to \langle b\rangle \to \langle a\rangle$. Such a graph is a relational structure over which we can evaluate the truth of the formula $\varphi_{bb}^{\mathrm{no}}$. Variables like u and v range over the nodes of the graph, $\rightharpoonup$ represents the edge relation, and the symbols $\langle a\rangle$ and $\langle b\rangle$ are to be interpreted as unary relations indicating that a node is labeled by an a and a b, respectively.

The first conjunct of $\varphi_{bb}^{\mathrm{no}}$ specifies that no two consecutive nodes are both labeled by a b, while the second conjunct ensures that the last node is labeled by an a.

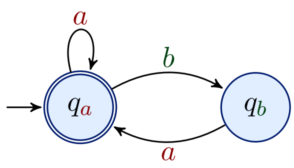


Figure 1.1. $\mathcal{A}_{bb}^{\text{no}}$, a nondeterministic finite automaton whose language, when restricted to nonempty words, consists of all the finite words over the alphabet $\Sigma = \{a, b\}$ that do not contain the segment bb and that end with an a.

In the formula φ_{bb}^{no} of the preceding example, we only used quantifiers that range over the nodes of a graph. By additionally allowing quantification over sets of nodes, we reach the full extent of *monadic second-order* (MSO) logic. The famous result established by Büchi, Elgot and Trakhtenbrot states that we can effectively translate every finite automaton to an equivalent MSO-formula (with relation symbols fixed as in Example 1.1.1), and vice versa. (For a proof, see, e.g., [Tho96, Thm 3.1].) An important consequence of this equivalence is that the satisfiability and validity problems of MSO-logic on words are decidable, because so are the corresponding problems for finite automata. This application was the original motivation for establishing a connection between the two worlds. Nowadays, this and similar 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, Wright [TW68] and Doner [Don70] (see, e.g., [Tho96, Thm 3.8]). The corresponding tree automata (which we shall not consider here) can be seen as a natural extension of finite automata to labeled trees. As far as MSO-logic is concerned, the generalization to (ordered, directed) labeled trees is straightforward, since, just like words, these can be regarded as labeled graphs. We only need to introduce additional edge relation symbols of the form $\xrightarrow{1}$, $\xrightarrow{2}$, ..., in order to be able to express that some node is the i-th child of another node.

In view of these results, it seems natural to ask whether the bridge between automata theory and logic persists if we expand our field of interest from words or trees to arbitrary node labeled graphs (possibly with multiple edge relations, as for trees). However, the trouble is that this question is not well-defined. While we can easily specify what we mean by MSO-logic on graphs, it is not obvious at all how finite automata should be canonically generalized to graphs that go beyond trees.

> *A result similar to those* [for word and tree languages] *does not exist for graph languages, for the trivial reason that there is no agreement on what would be the class of "regular graph languages", and, in particular, that there is no accepted notion of "finite graph automaton".*
>
> (Joost Engelfriet, 1991 [Eng91, p. 139])

Nevertheless, graph languages have been an active area of research for nearly fifty years. In large part, this has been driven by investigations of generative devices known as *graph grammars*. The theory of graph grammars is by now well-developed, as can be seen from the "Handbook of Graph Grammars and Computing by Graph Transformation" [HGG97-99], a comprehensive reference consisting of several volumes. Within this branch of research, MSO-logic has raised considerable interest. Especially through the work of Courcelle, MSO-formulas have proven to be particularly useful tools for obtaining decidability results about graph languages.

> *The deep reason why* [monadic second-order] *logic is so crucial is that it replaces for graphs* [...] *the notion of a finite automaton which is very important in the theory of formal languages. It "replaces" because no convenient notion of finite automaton is known for graphs.*
>
> (Bruno Courcelle, 1997 [Cou97, p. 315])

Supported by the equivalence of regularity and MSO-definability on words and labeled trees, one might even go as far as referring to the sets of graphs that can be defined by MSO-formulas as the "regular graph languages". Hence, one way to cope with the lack of a canonical generalization of finite automata to graphs is to search for a model of computation on graphs that has the same expressive power as MSO-logic. This is the approach pursued in this thesis.

It must be emphasized that the present work is not, by any means, the first to investigate graph automata. The definitions suggested in the literature over the last decades are far too numerous to survey here, but let us at least mention a small selection. Already in the early days of graph grammars, mostly in the 1970s, the notion of graph-accepting machines was explored in parallel to generative devices. Some examples, among many others, are the models proposed by Shah, Milgram, Wu and Rosenfeld in [SMR73], [Mil75] and [WR79]. However, none of those studies were concerned with equivalence to MSO-logic, and few of them were pursued much further. It seems that graph grammars received much more interest than graph automata. Later, in the early 1990s, Thomas introduced his *graph acceptors* in [Tho91], with the explicit goal of an automata-theoretic investigation of MSO-definable graph properties. It turned out that Thomas' graph acceptors recognize precisely the graph languages of bounded degree definable in the existential fragment(1) of MSO-logic (see [Tho97, Thm 3]). This makes them less expressive than full MSO-logic, and, in particular, their class of recognizable languages is not closed under complementation. Also, about a year earlier, Courcelle had introduced in [Cou90] an algebraic notion of *recognizability*, without defining any notion of graph automaton. Every MSO-definable graph language is recognizable in Courcelle's sense, but not vice versa.

The expressiveness of MSO-logic on graphs has thus been approximated "from below", by Thomas, and "from above", by Courcelle, but, to the author's best knowledge, a perfectly matching automaton model has been missing so far. Relatively recent remarks by Courcelle and Engelfriet in [CE12], as well as the following explicit statement, support this assumption.

> *No existing notion of graph automaton gives an equivalence with monadic second-order logic.*
>
> (Bruno Courcelle, 2008 [Cou08, p. 8])

(1) The existential fragment of MSO-logic consists of formulas of the form $\exists \mathsf{U}_1, \dots, \mathsf{U}_n(\varphi)$, where $\mathsf{U}_1, \dots, \mathsf{U}_n$ are set variables, and φ is a first-order formula (i.e., φ does not contain any set quantifiers).

The present work is an attempt to close this gap in the theory of graph languages. It is successful in the sense that it provides a class of graph automata equivalent to MSO-logic. However, it must also be conceded that, up to now, no new results have been inferred from this alternative characterization. Whether it will prove as fruitful as classical automata on words and trees remains to be seen.

1.2 Structure of this Thesis

The presentation is organized as follows: After some preliminaries on graphs in Chapter 2, we introduce the alternating variant of our distributed graph automata (ADGAs) in Chapter 3, and discuss some of their properties. Since the capabilities of ADGAs might not be obvious at first sight, a substantial part of the chapter is devoted to examples. Then, in Chapter 4, we review MSO-logic on graphs, and prove our main result, the equivalence of MSO-logic and ADGAs. This immediately entails some negative results on ADGAs. We finish by considering nondeterministic and deterministic variants of our automata in Chapter 5. Both turn out to be strictly weaker than ADGAs, and they also form a hierarchy among themselves. The loss of expressive power is however rewarded by a decidable emptiness problem.

Chapter 2

Preliminaries on Graphs

Graphs play a central role in this work. On the one hand, they will serve as input for automata, and as models for logical formulas. On the other hand, we will use them to describe the behaviour of automata, and to represent two-player games. In this chapter, we provide formal definitions, review some common graph properties, and discuss the notion of graph minors.

2.1 Basic Definitions

As our most general concept, we consider directed graphs with nodes labeled by an alphabet Σ, and multiple edge relations indexed by an alphabet Γ. While node labels are auxiliary, we regard edge labels as an integral part of the graph structure.

2.1.1 Definition (Γ-Graph).
Let Γ be a nonempty finite alphabet (i.e., a set of symbols). A Γ-graph G is a structure $\big\langle V_G, \langle \xrightarrow{\gamma}_G \rangle_{\gamma \in \Gamma} \big\rangle$, where

- V_G is a nonempty finite set of nodes, and
- each $\xrightarrow{\gamma}_G \subseteq V_G \times V_G$ is a set of directed edges labeled by $\gamma \in \Gamma$.

If Γ is understood or irrelevant, we refer to G simply as a graph. Note that self-loops are allowed, and that there can be multiple edges from one node to another, but at most one for every edge label $\gamma \in \Gamma$. If there are no self-loops, i.e., if $v \xrightarrow{\gamma}_G v$ does not hold for any $v \in V_G$ and $\gamma \in \Gamma$, then we say that G is loop-free. Furthermore, if there is only a single edge relation, we call G a simple graph. In such a case, we set $\Gamma = \{\square\}$ (singleton consisting of a blank symbol), and omit the superfluous edge labels.

2.1.2 Definition (Σ-Labeled Γ-Graph).
Let Σ and Γ be two nonempty finite alphabets. A Σ-labeled Γ-graph is a tuple $\langle G, \lambda \rangle$, denoted as G_λ, where

- G is a Γ-graph (referred to as the underlying graph), and
- $\lambda \colon V_G \to \Sigma$ is a node labeling (function).

Again, we will often relax our terminology, and refer to G_λ simply as a labeled graph, or even as a graph, if the meaning is clear from the context.

At this point, it is important to mention that we are only interested in (labeled) graphs *up to isomorphism*. That is, we consider two Σ-labeled Γ-graphs G_λ and $G'_{\lambda'}$ to be equal if there is a bijection $f\colon V_G \to V_{G'}$, such that $\lambda(v) = \lambda'(f(v))$, and $u \xrightarrow{\gamma}_G v$ if and only if $f(u) \xrightarrow{\gamma}_{G'} f(v)$, for all $u, v \in V_G$ and $\gamma \in \Gamma$. The reason for this is that our automata and logical formulas cannot distinguish between isomorphic graphs.

In order to draw an analogy to formal language theory on words, we introduce an extension to graphs of the well-known notation employing alphabet exponentiation and the Kleene star. In the context of words, Σ^n designates the set of all words of length n over the alphabet Σ, and Σ^* the set of all words of arbitrary (finite) length over Σ. Now, a word over Σ can be viewed as a Σ-labeled linear graph with a single edge relation. For example, we can identify $acab$ with $a \to c \to a \to b$. From this point of view, the number n in the expression Σ^n refers to the underlying linear graph of length n, and the Kleene star $*$ in Σ^* can be seen as a placeholder for any linear graph. We can generalize this notation to arbitrary finite graphs by replacing n with a Γ-graph G, and $*$ with the symbol $\textcircled{$\Gamma$}$ ("clouded Γ"), which serves as a placeholder for any Γ-graph.

2.1.3 Definition (Cloud Notation).
For any nonempty finite alphabets Σ and Γ, and any Γ-graph G, we denote by Σ^G the set of all Σ-labeled graphs with underlying graph G, and by $\Sigma^{\textcircled{$\Gamma$}}$ the set of all Σ-labeled Γ-graphs, i.e.,

$$\Sigma^G := \{G_\lambda \mid \lambda\colon V_G \to \Sigma\}, \quad \text{and} \quad \Sigma^{\textcircled{Γ}} := \bigcup_{G \in \mathcal{G}(\Gamma)} \Sigma^G,$$

where $\mathcal{G}(\Gamma)$ is the set of all Γ-graphs.

Occasionally, the need will arise to consider graphs labeled heterogeneously with two alphabets Σ_1 and Σ_2, such that at least one node is labeled by some symbol from Σ_1 that is not contained in Σ_2. In such cases, we shall employ the notational shorthands

$$\begin{aligned} \langle \Sigma_1, \Sigma_2 \rangle^G &:= (\Sigma_1 \cup \Sigma_2)^G \setminus (\Sigma_2)^G, \quad \text{and} \\ \langle \Sigma_1, \Sigma_2 \rangle^{\textcircled{Γ}} &:= (\Sigma_1 \cup \Sigma_2)^{\textcircled{Γ}} \setminus (\Sigma_2)^{\textcircled{Γ}}. \end{aligned}$$

We will use automata and logical formulas to characterize sets of labeled graphs. Pursuing the analogy with words, such sets are called graph languages.

2.1.4 Definition (Graph Language).
A graph language is a set of labeled graphs. More precisely, L is a graph language if and only if there are finite alphabets Σ and Γ, such that $L \subseteq \Sigma^{\textcircled{Γ}}$.

In many examples, we will consider graph languages for which the node labels are irrelevant. In such cases, we fix the node alphabet to be a singleton $\Sigma = \{\square\}$,[1] and, to simplify notation, we identify any labeled graph $G_\lambda \in \Sigma^{\textcircled{Γ}}$

[1] Since Σ and Γ do not need to be disjoint, we can use the same blank symbol $\square$ for both.

with its underlying graph G.

Furthermore, when reasoning about graphs as structural objects, we will follow the usual terminology of graph theory. In particular, given a Γ-graph G and two nodes $u, v \in V_G$, we say that u is an incoming neighbor of v, and v an outgoing neighbor of u, if $u \xrightarrow{\gamma}_G v$ for some $\gamma \in \Gamma$. A node without incoming neighbors is called a source, and a node without outgoing neighbors a sink. 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. Accordingly, the incoming and outgoing neighborhoods contain only the incoming and outgoing neighbors, respectively. If we additionally qualify a neighborhood of a node v as closed, it means that we also include v itself into the set.

A (directed) path from u to v is a sequence of nodes, starting with u and ending with v, in which each node but the last is (directly) followed by one of its outgoing neighbors. If a subsequent neighbor does not necessarily have to be outgoing, we call the sequence an undirected path.

We say that a Γ-graph H is a subgraph of another Γ-graph G (or that G contains H as a subgraph) if $V_H \subseteq V_G$ and $\xrightarrow{\gamma}_H \subseteq (\xrightarrow{\gamma}_G \cap V_H^2)$ for all $\gamma \in \Gamma$. If a subgraph H contains all the edges between nodes in V_H that occur in G, i.e., if $\xrightarrow{\gamma}_H = (\xrightarrow{\gamma}_G \cap V_H^2)$ for all $\gamma \in \Gamma$, then we call H the subgraph of G induced by V_H, and denote it by $G[V_H]$.

2.2 Some Graph Properties

We now briefly recall some standard graph properties, which will serve us as examples of graph languages in Chapters 3 and 4. Let Σ and Γ be two nonempty finite alphabets, and G_λ some Σ-labeled Γ-graph. In the following, if node labels are irrelevant for some graph property, we only refer to the underlying graph G, but, of course, the same properties also apply to labeled graphs.

The node labeling λ constitutes a valid coloring of G if no two adjacent nodes (neighbors) share the same label, i.e., $u \xrightarrow{\gamma}_G v$ implies $\lambda(u) \neq \lambda(v)$, for all $u, v \in V_G$ and $\gamma \in \Gamma$. If $|\Sigma| = k$, such a labeling λ is called a k-coloring of G, and any Γ-graph for which a k-coloring exists is said to be k-colorable. Note that, by definition, a graph that contains self-loops is not k-colorable for any $k \geq 1$.

We call G (weakly) connected if there is an undirected path between every two nodes $u, v \in V_G$.

If we want to deal with undirected graphs, we can represent them as directed graphs with bidirectional edges. Formally, G is undirected if for every $u, v \in V_G$ and $\gamma \in \Gamma$, it holds that $u \xrightarrow{\gamma}_G v$ if and only if $v \xrightarrow{\gamma}_G u$. For undirected graphs, we shall not distinguish between the γ-edge from u to v and the γ-edge from v to u. Instead, we refer collectively to both of them as *the* γ-edge between u and v.

A (simple) cycle in G is a path that starts and ends at the same node $v \in V_G$, such that v occurs precisely two times, and every other node in V_G occurs at most once. If such a cycle contains every node in V_G, it is called a Hamiltonian cycle.

We say that a set of edges $M \subseteq \bigcup_{\gamma \in \Gamma}(\xrightarrow{\gamma}_G)$ is a perfect matching of G if no two edges in M share a common node, and every node in V_G is covered by some edge in M.

A nontrivial automorphism of G_λ is a bijection $f\colon V_G \to V_G$ that is not an identity, such that $\lambda(v) = \lambda(f(v))$, and $u \xrightarrow{\gamma}_G v$ if and only if $f(u) \xrightarrow{\gamma}_G f(v)$, for all $u, v \in V_G$ and $\gamma \in \Gamma$.

The graph G is planar if it can be drawn in the plane such that no two edges intersect each other, i.e., they may only meet at nodes. We will give a more formal characterization of planarity in the next section (see Theorem 2.3.5).

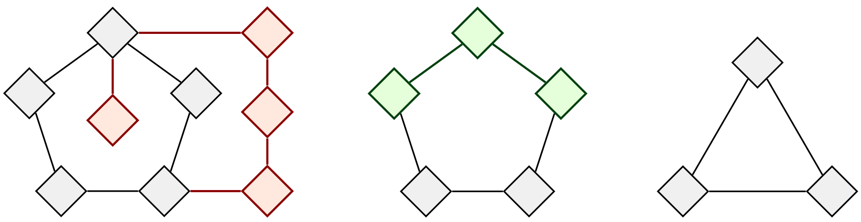
2.3 Graph Minors

The issue with the definition of a planar graph given above is that it refers to the notion of a plane, an object that is not part of the graph. In order to make planarity accessible to our automata, we need a specification that involves only the graph structure itself. We will exploit an important result in graph theory for this purpose: the characterization of planarity in terms of forbidden minors. In this context, we only consider unlabeled, simple, undirected graphs.

Graph minors can be defined by means of edge contractions. Given a simple undirected graph G, contracting the edge between two nodes $u, v \in V_G$ means to remove that edge and merge the nodes u and v, such that every node that was a neighbor of u or v becomes a neighbor of the merged node.

2.3.1 Definition (Minor).
Let G and H be two simple undirected graphs, such that H is loop-free. We say that H is a minor of G (or that G contains H as a minor) if we can obtain H by taking a subgraph of G and repeatedly contracting edges.

2.3.2 Example.
Consider the graph in Fig. 2.1a. Removing the nodes and edges highlighted in red yields the subgraph shown in Fig. 2.1b. Then, by contracting the edges highlighted in green, we obtain K_3, the complete graph with three nodes depicted in Fig. 2.1c. Hence, the considered graph contains K_3 as a minor.



(a) considered graph **(b)** subgraph **(c)** K_3

Figure 2.1. A graph that contains the complete graph with three nodes (K_3) as a minor.

In the preceding example, we could obtain K_3 as a minor because the considered graph contains a sufficiently large cycle. It is easy to see that this is a necessary and sufficient condition.

2.3.3 Remark.
A simple undirected graph contains K_3 as a minor if and only if it contains at least one cycle of three or more nodes.

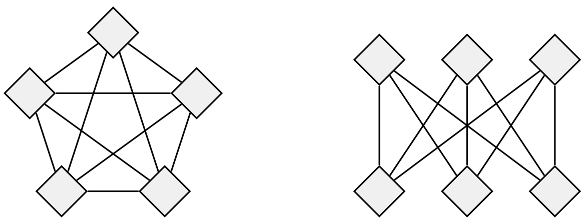
While the notion of contracting edges is intuitive, it is not directly available in the logical and automata-theoretic formalisms that we will consider. Instead, we will use the following characterization of minor inclusion given by Courcelle and Engelfriet in [CE12, Lemma 1.13].

2.3.4 Lemma (Minor Inclusion).
Let G and H be two simple undirected graphs, such that H is loop-free and $V_H = \{v_1, \dots, v_n\}$. Then H is a minor of G if and only if there exist pairwise disjoint nonempty sets of nodes $U_1, \dots, U_n \subseteq V_G$, such that

- each induced subgraph $G[U_i]$ is connected, for $1 \leq i \leq n$, and
- for every edge in H between two nodes $v_i, v_j \in V_H$, there exists an edge in G between two nodes $u, v \in V_G$ such that $u \in U_i$ and $v \in U_j$.

We now come back to planarity. An important theorem by Kuratowski [Kur30] characterizes the planar graphs in terms of two forbidden graphs: the complete graph with five nodes K_5, and the complete bipartite graph with two times three nodes $\mathrm{K}_{3,3}$, both depicted in Fig. 2.2. In Wagner's variant of the theorem [Wag37], which we shall use, those forbidden graphs may not occur as minors (for a proof, see, e.g., [Die10, Thm 4.4.6]).

2.3.5 Theorem (Kuratowski-Wagner Theorem).
A simple undirected graph is planar if and only if it contains *neither* K_5 *nor* $\mathrm{K}_{3,3}$ as a minor.



(a) K_5 **(b)** $\mathrm{K}_{3,3}$

Figure 2.2. The complete graph with five nodes K_5, and the complete bipartite graph with two times three nodes $\mathrm{K}_{3,3}$.

Chapter 3

Alternating Distributed Graph Automata

In this chapter, we introduce the main variant of distributed graph automata investigated in this work, and examine some of their properties. In particular, we establish a game-theoretic characterization of their acceptance condition, and derive some closure properties of their class of recognizable languages, both of which will be useful for proving our main result in Chapter 4.

3.1 Preview

We start with an informal description of the automaton model. Formal definitions follow in Section 3.2.

An alternating distributed graph automaton (ADGA) is an abstract machine that, given a labeled graph as input, can either accept or reject it, thereby specifying a graph language. Our model of computation incorporates the following key concepts:

Synchronous Distributed Algorithm.
An ADGA operates primarily as a distributed algorithm. Each node of the input graph is assigned its own local processor, which we shall not distinguish from the node itself. Communication takes place in synchronous rounds, in which each node receives the current states of its incoming neighbors.

Finite-State Machines.
Each local processor is a finite-state machine, i.e., an abstract machine that can be in one of a finite number of states, and has no additional memory. Its initial state is determined by the node label. After each communication round, it updates its state according to a nondeterministic transition function that depends only on the current state and the states received from the incoming neighborhood.

Constant Running Time.
The number of communication rounds is limited by a constant. To ensure this, 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 graph to terminate too.

Aggregation of States.
In order to be finitely representable, an ADGA treats collections of states as sets, i.e., it abstracts away from the multiplicity of states. This aggregation of states into sets is applied in two ways:

- First, the information received by the nodes in each round is a family of sets of states, indexed by the edge alphabet of the graph. That is, for each edge relation, a node knows which states occur in its incoming neighborhood, but it cannot distinguish between neighbors that are in the same state.
- Second, once all the nodes have reached a permanent state, the ADGA 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 is part of the ADGA's accepting sets, then the input graph is accepted, otherwise it is rejected.

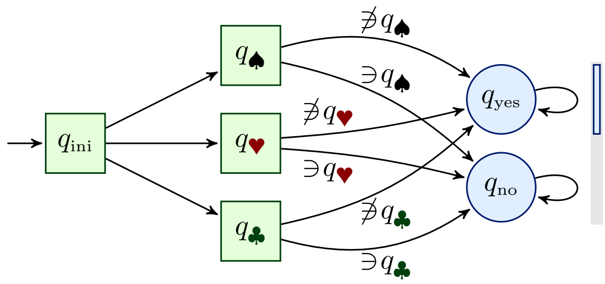


Figure 3.1. $\mathcal{A}_3^{\mathrm{color}}$, an ADGA over $\langle\{\square\},\{\square\}\rangle$ whose graph language consists of the 3-colorable graphs.

3.1.1 Example (3-Colorability).
Figure 3.1 shows a simple ADGA $\mathcal{A}_3^{\mathrm{color}}$, represented as a state diagram. The states are arranged in columns corresponding to their levels, ascending from left to right. $\mathcal{A}_3^{\mathrm{color}}$ expects a $\{\square\}$-labeled $\{\square\}$-graph as input, and accepts it if and only if it is 3-colorable. The automaton proceeds as follows: All nodes of the input graph are initialized to the state q_{ini}. In the first round, each node nondeterministically chooses to go to one of the states $q_{\spadesuit}$, $q_{\heartsuit}$ and $q_{\clubsuit}$, 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 graph 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 blue bar to the right of the state diagram. We shall refer to such a representation of sets using bars as *barcode*.

One last key concept that enters into the definition of ADGAs is *alternation*, a generalization of nondeterminism introduced by Chandra, Kozen and Stockmeyer in [CKS81] (in their case, for Turing machines and other types of word automata).

Alternating Automaton.

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 ADGA 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 ADGA 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 ADGA then accepts the input graph if and only if its acceptance condition is satisfied in every universal branch of the run.

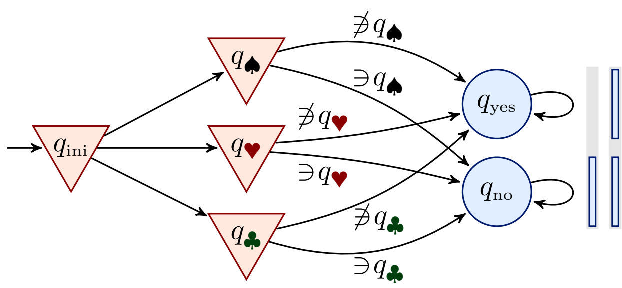


Figure 3.2. $\bar{\mathcal{A}}_3^{\text{color}}$, an ADGA over $\langle\{\square\},\{\square\}\rangle$ whose graph language consists of the graphs that are *not* 3-colorable.

3.1.2 Example (Non-3-Colorability).
To illustrate the notion of universal branching, consider the ADGA $\bar{\mathcal{A}}_3^{\text{color}}$ shown in Fig. 3.2. It is a complement automaton of $\mathcal{A}_3^{\text{color}}$ from Example 3.1.1, i.e., it accepts precisely those $\{\square\}$-labeled $\{\square\}$-graphs that are *not* 3-colorable. States represented as red triangles are universal (whereas the green squares in Fig. 3.1 stand for existential states). Given an input graph with n nodes, $\bar{\mathcal{A}}_3^{\text{color}}$ 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 $\mathcal{A}_3^{\text{color}}$ running on the same input graph. 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 $\bar{\mathcal{A}}_3^{\text{color}}$ is satisfied if and only if at least one node is in state q_{no}, i.e., the accepting sets are $\{q_{\text{no}}\}$ and $\{q_{\text{yes}}, q_{\text{no}}\}$. Hence, the automaton accepts the input graph if and only if no valid coloring was found in any universal branch. Note that we could also have chosen to make the states $q_{\spadesuit}$, $q_{\heartsuit}$ and $q_{\clubsuit}$ 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.

3.2.1 Definition (Alternating Distributed Graph Automaton).
An alternating distributed graph automaton (ADGA) $\mathcal{A}$ over $\langle \Sigma, \Gamma \rangle$ is a tuple $\langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$, where

- Σ is a finite nonempty alphabet of node labels,
- Γ is a finite alphabet of edge labels,
- $\widehat{Q} = \langle Q_\exists, Q_\forall, Q_\mathrm{P} \rangle$, where $Q_\exists$, $Q_\forall$ and Q_P are pairwise disjoint finite sets of existential states, universal states and permanent states, respectively, with $Q_\mathrm{P} \neq \emptyset$, and for notational convenience we use the abbreviations
  - $Q \coloneqq Q_\exists \cup Q_\forall \cup Q_\mathrm{P}$, for the entire set of states, and
  - $Q_\mathrm{N} \coloneqq Q_\exists \cup Q_\forall$, for the set of nonpermanent states,
- $\sigma \colon \Sigma \to Q$ is an initialization function,
- $\delta \colon Q \times (2^Q)^\Gamma \to 2^Q$ is a (local) transition function that allows to unambiguously associate a level $\mathrm{l}_\mathcal{A}(q) \in \mathbb{N}$ with every state $q \in Q$, such that
  - transitions between nonpermanent states go only from one level to the next, where the lowest level consists of the nonpermanent states that have no incoming transitions, which are also the only nonpermanent states that can be assigned by the initialization function, i.e., for every $q \in Q_\mathrm{N}$,

$$\mathrm{l}_\mathcal{A}(q) = \begin{cases} 0 & \text{if for all } p \in Q \text{ and } \widehat{S} \in (2^Q)^\Gamma, \text{ it holds that } \\ & q \notin \delta(p, \widehat{S}), \\ i+1 & \text{if there are } p \in Q_\mathrm{N} \text{ and } \widehat{S} \in (2^Q)^\Gamma \text{ with} \\ & \mathrm{l}_\mathcal{A}(p) = i \text{ and } q \in \delta(p, \widehat{S}), \end{cases}$$

$$\exists a \in \Sigma \colon \sigma(a) = q \quad \text{implies} \quad \mathrm{l}_\mathcal{A}(q) = 0,$$

  - the permanent states are one level higher than the highest nonpermanent ones, and have only self-loops as outgoing transitions, i.e., for every $q \in Q_\mathrm{P}$,

$$\mathrm{l}_\mathcal{A}(q) = \begin{cases} 0 & \text{if } \ Q_\mathrm{N} = \emptyset, \\ \max\{\mathrm{l}_\mathcal{A}(q) \mid q \in Q_\mathrm{N}\} + 1 & \text{otherwise,} \end{cases}$$

$$\delta(q, \widehat{S}) = \{q\} \quad \text{for every } \widehat{S} \in (2^Q)^\Gamma,$$

  - states on the same level are in the same component of $\widehat{Q}$, i.e., for every level $i \in \mathbb{N}$, $\{q \in Q \mid \mathrm{l}_\mathcal{A}(q) = i\} \in (2^{Q_\exists} \cup 2^{Q_\forall} \cup 2^{Q_\mathrm{P}})$, and
- $\mathcal{F} \subseteq 2^{Q_\mathrm{P}}$ is a set of accepting sets of permanent states.

When specifying an ADGA formally, we only need to indicate the outgoing transitions of the nonpermanent states, since permanent states are, by definition, self-looping. In many cases, however, we will opt for a more convenient specification through a state diagram, as we already did in Section 3.1. Let us clarify the details of such a representation by means of a slightly more involved example.

3.2.2 Example (ADGA Specification through a State Diagram).
Consider the ADGA $\mathcal{A}_{\text{centric}} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ shown in Fig. 3.3. (For now, we are not interested in the graph language that it recognizes. This will be discussed later in Example 3.2.8.) As indicated by the caption, $\Sigma = \{a, b, c\}$, and $\Gamma = \{\square\}$. Existential states are represented by green squares, universal states by red triangles, and permanent states by blue circles. The short arrows mapping node labels to states indicate the initialization function. For instance, $\sigma(a) = q_a$.

As usual, the arrows between states specify the transition function. A label on such a transition arrow indicates a condition on the states in the incoming neighborhood of a node, which must be satisfied in order for the node to be allowed to take that transition. If there is no label, any states in the neighborhood are permitted. In this example (and any other example that we shall consider) the permitted input graphs have a single edge relation, which means that the occurrences of states the incoming neighborhood of a node are abstracted as a single set of states S. Now, the labels on the transition arrows are formulas that specify conditions on such a set S. In those formulas, S remains anonymous, and binary relations like $\in$ and $=$ are written as if they were unary, but implicitly refer to S. For example, "$\ni q_b$" means "$q_b \in S$", and "$=\{q_{b\clubsuit}, q_{b\blacklozenge}\}$" means "$S = \{q_{b\clubsuit}, q_{b\blacklozenge}\}$". Consequently, considering the three outgoing arrows of q_b tells us, for instance, that $\delta\big(q_b, \langle\{q_a, q_c\}\rangle\big) = \{q_{b\clubsuit}, q_{b\blacklozenge}\}$ and $\delta\big(q_b, \langle\{q_a, q_b\}\rangle\big) = \{q_{\text{no}}\}$. Furthermore, we can build up more complex formulas using the usual boolean connectives $\neg$, $\vee$, $\wedge$, etc. Hence, "$\not\ni q_c \wedge \not\ni q_a$" characterizes all the sets of states that contain neither q_c nor q_a. We shall refer to such formulas as *set formulas*.

Finally, as already mentioned in Section 3.1, the *barcode* on the far right specifies the accepting sets. The blue bars are aligned with the permanent states to which they correspond. Each column represents an accepting set, where a bar means that the corresponding permanent state is included in the set. Thus, $\mathcal{F} = \big\{\{q_{a\spadesuit}, q_{\text{yes}}\}, \{q_{a\heartsuit}, q_{\text{yes}}\}\big\}$.

For any ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$, we define its size $\operatorname{siz}(\mathcal{A})$ to be its number of states, i.e., $\operatorname{siz}(\mathcal{A}) := |Q|$, and its length $\operatorname{len}(\mathcal{A})$ to be its highest level, i.e., $\operatorname{len}(\mathcal{A}) := \max\{\mathrm{l}_{\mathcal{A}}(q) \mid q \in Q\}$. For example, we get $\operatorname{siz}(\mathcal{A}_{\text{centric}}) = 10$ and $\operatorname{len}(\mathcal{A}_{\text{centric}}) = 2$, for the automaton from Fig. 3.3.

As a further abbreviation, we shall use $\operatorname{lev}_i(\mathcal{A}) := \{q \in Q \mid \mathrm{l}_{\mathcal{A}}(q) = i\}$, for the set of states at level i, with $0 \le i \le \operatorname{len}(\mathcal{A})$. For instance, $\operatorname{lev}_1(\mathcal{A}_{\text{centric}}) = \{q'_a, q_{b\clubsuit}, q_{b\blacklozenge}\}$. We say that level i of $\mathcal{A}$ is existential if $\operatorname{lev}_i(\mathcal{A}) \subseteq Q_\exists$, and analogously for universal and permanent levels. For $\mathcal{A}_{\text{centric}}$, level 0 is existential, level 1 is universal, and level 2 is permanent.

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 ADGA.

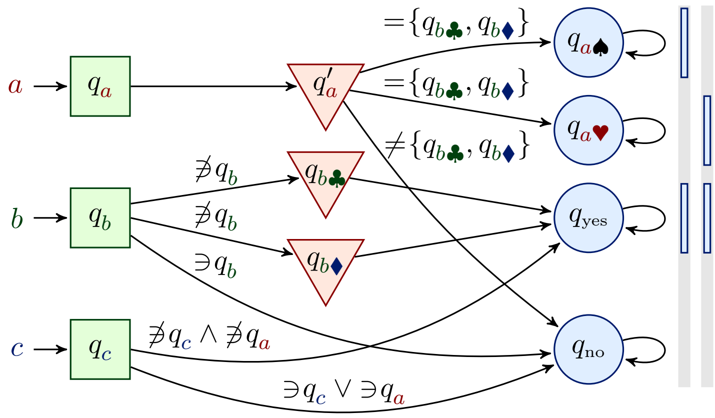


Figure 3.3. $\mathcal{A}_{\text{centric}}$, an ADGA over $\langle\{a, b, c\}, \{\square\}\rangle$ whose graph language consists of the labeled graphs that satisfy the following conditions: the labeling constitutes a valid 3-coloring, there is precisely one a-labeled node v_a, the undirected neighborhood of v_a contains only b-labeled nodes, and v_a has at least two incoming neighbors.

3.2.3 Definition (Configuration).
Consider an ADGA $\mathcal{A} = \langle \Sigma, \varGamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$. We call any Q-labeled $\varGamma$-graph $G_\kappa \in Q^{\textcircled{\varGamma}}$ a configuration of $\mathcal{A}$ on G. If every node is labeled by a permanent state, i.e., if $G_\kappa \in (Q_{\mathrm{P}})^{\textcircled{\varGamma}}$, we call G_κ a permanent configuration. Otherwise, if G_κ is a nonpermanent configuration whose nodes are labeled exclusively by existential and permanent states, i.e., if $G_\kappa \in \langle Q_\exists, Q_{\mathrm{P}}\rangle^{\textcircled{\varGamma}}$, we say that G_κ is an existential configuration. Analogously, if $G_\kappa \in \langle Q_\forall, Q_{\mathrm{P}}\rangle^{\textcircled{\varGamma}}$, the configuration is called universal.

Additionally, we say that a permanent configuration G_κ is accepting if the set of states occurring in it is accepting, i.e., if $\{\kappa(v) \mid v \in V_G\} \in \mathcal{F}$. Any other permanent configuration is called rejecting. Nonpermanent configurations are neither accepting nor rejecting.

The (local) transition function of an ADGA 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.

3.2.4 Definition (Global Transition Function).
The global transition function $\delta^{\circledast}$ of an ADGA $\mathcal{A} = \langle \Sigma, \varGamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$ assigns to each configuration G_κ of $\mathcal{A}$ the set of all of its successor configurations G_μ, by combining all possible outcomes of local transitions on G_κ, i.e.,

$$
\begin{aligned}
\delta^{\circledast}\colon Q^{\textcircled{\varGamma}} &\to 2^{(Q^{\textcircled{\varGamma}})} \\
G_\kappa &\mapsto \Big\{ G_\mu \;\Big|\; \bigwedge_{v \in V_G} \mu(v) \in \delta\Big(\kappa(v),\, \big\langle \{\kappa(u) \mid u \xrightarrow{\gamma}_G v\}\big\rangle_{\gamma\in\varGamma}\Big)\Big\}.
\end{aligned}
$$

A configuration G_μ that can be obtained from G_κ by iteratively choosing some successor configuration shall be referred to as a descendant configuration of G_κ.

We now have everything at hand to formalize the notion of a run. As mentioned in Section 3.1, a run can be split into several parallel branches whenever the nodes of the input graph are in universal states. It thus may seem natural to define a run as a tree whose nodes are labeled by configurations of the automaton. We could then interpret the branches of such a tree as "non-communicating parallel universes". However, since an ADGA has no "memory of the past" other than its current configuration, there is no need to keep apart branches that are in the same configuration in a given round. By merging such branches, we obtain a directed acyclic graph in which every node is labeled by a *unique* configuration (since a configuration cannot occur in more than one round). This has the advantage that we can identify the nodes of a run with the configurations of an automaton, which will make it easier to refer to particular nodes and paths of a run in subsequent proofs.

3.2.5 Definition (Run).
A run of an ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ on a labeled graph $G_\lambda \in \Sigma^{\circledast_{\Gamma}}$ is a directed acyclic graph $R = \langle K, \rightharpoonup \rangle$ whose nodes are configurations of $\mathcal{A}$ on G, i.e., $K \subseteq Q^G$, such that

- the initial configuration $G_{\sigma \circ \lambda} \in K$ is the only source,[(1)]
- every existential configuration $G_\kappa \in (K \cap \langle Q_\exists, Q_\mathrm{P} \rangle^G)$ has exactly one outgoing neighbor $G_\mu \in \delta^{\circledast}(G_\kappa)$,
- every universal configuration $G_\kappa \in (K \cap \langle Q_\forall, Q_\mathrm{P} \rangle^G)$ with $\delta^{\circledast}(G_\kappa) = \{G_{\mu_1}, \ldots, G_{\mu_m}\}$ has exactly m outgoing neighbors $G_{\mu_1}, \ldots, G_{\mu_m}$, and
- every permanent configuration $G_\kappa \in (K \cap Q_\mathrm{P}^G)$ is a sink.

Such a run R is accepting if every occurring permanent configuration is accepting, i.e., if K contains a node $G_\kappa \in Q_\mathrm{P}^G$, then $\{\kappa(v) \mid v \in V_G\} \in \mathcal{F}$. Otherwise, R is called rejecting.

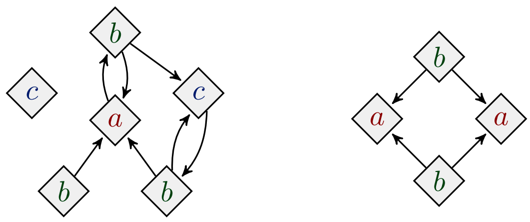
3.2.6 Example (Two Runs of $\mathcal{A}_{\text{centric}}$).
We take up Example 3.2.2, and consider again the ADGA $\mathcal{A}_{\text{centric}}$ from Fig. 3.3.

The graph in Fig. 3.5 is a run of $\mathcal{A}_{\text{centric}}$ on the labeled graph $(G_1)_{\lambda_1}$ shown in Fig. 3.4a. (The figures are also depicted together on the cover of this thesis.) We have adopted the same coloring scheme as for (automaton) states, i.e., a green configuration is existential, a red one is universal, and a blue one is permanent. In the first round, the three nodes that are in state q_b have a nondeterministic choice between $q_{b\clubsuit}$ and $q_{b\blacklozenge}$. 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 $q_{a\spadesuit}$ and $q_{a\heartsuit}$. In both branches, an accepting configuration is reached, since $\{q_{a\spadesuit}, q_{\text{yes}}\}$ and $\{q_{a\heartsuit}, q_{\text{yes}}\}$ are both accepting sets. This is also visually indicated by the double circles around the configurations. We conclude that the run is accepting.

As an example of a rejecting run, consider Fig. 3.6 which shows a run

(1) Here, the operator $\circ$ denotes function composition, such that $(\sigma \circ \lambda)(v) = \sigma(\lambda(v))$.

of the same automaton $\mathcal{A}_{\text{centric}}$ on the labeled graph $(G_2)_{\lambda_2}$ from Fig. 3.4b. Again, the configuration chosen during the first round is one of several (four) possibilities. In the second round, the run is split into four universal branches, corresponding to the four possible combinations of choices of the two nodes that are in state q'_a. The permanent configurations reached in the two middle branches are rejecting because $\{q_{a\spadesuit}, q_{a\heartsuit}, q_{\text{yes}}\}$ is not an accepting set of $\mathcal{A}_{\text{centric}}$. The occurrence of these rejecting configurations causes the entire run to be rejecting.



(a) $(G_1)_{\lambda_1}$ **(b)** $(G_2)_{\lambda_2}$

Figure 3.4. Two $\{a, b, c\}$-labeled $\{\square\}$-graphs.

For any ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ and labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$, a configuration $G_\kappa \in Q^{\textcircled{\Gamma}}$ is called reachable by $\mathcal{A}$ on G_λ if either $G_\kappa = G_{\sigma\circ\lambda}$ or there is a configuration $G_\mu \in Q^{\textcircled{\Gamma}}$ reachable by $\mathcal{A}$ on G_λ such that $G_\kappa \in \delta^{\pentagon}(G_\mu)$. If G_λ is irrelevant, we simply say that G_κ is reachable by $\mathcal{A}$. Note that only existential, universal and permanent configurations can satisfy this property, i.e., "mixed" configurations with both existential and universal states are never reachable. Furthermore, any reachable existential configuration $G_\kappa \in \langle Q_\exists, Q_\mathrm{P} \rangle^{\textcircled{\Gamma}}$ has existential states of uniform level, i.e., there is a level i, such that $\mathrm{l}_\mathcal{A}(\kappa(v)) = i$ for all nodes $v \in V_G$ with $\kappa(v) \in Q_\exists$. The analogous observation holds if G_κ is a reachable universal configuration.

In the following definition, we transfer the usual terminology of automata theory to ADGAs.

3.2.7 Definition (ADGA-Recognizability).
Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be an ADGA. A labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ is accepted by $\mathcal{A}$ if and only if there exists an accepting run R of $\mathcal{A}$ on G_λ. The graph language recognized by $\mathcal{A}$ is the set

$$\mathrm{L}(\mathcal{A}) := \{ G_\lambda \in \Sigma^{\textcircled{\Gamma}} \mid \mathcal{A} \text{ accepts } G_\lambda \}.$$

Every graph language that is recognized by some ADGA is called ADGA-recognizable. We denote by $\mathcal{L}_{\text{ADGA}}$ the class of all such graph languages.

If two ADGAs recognize the same graph language, we say that they are equivalent.

3.2.8 Example (Language Recognized by $\mathcal{A}_{\text{centric}}$).
We get back to the example automaton $\mathcal{A}_{\text{centric}}$ from Fig. 3.3, this time turning our attention to the graph language that it recognizes.

In the first round, the a-labeled nodes do nothing but update their state,

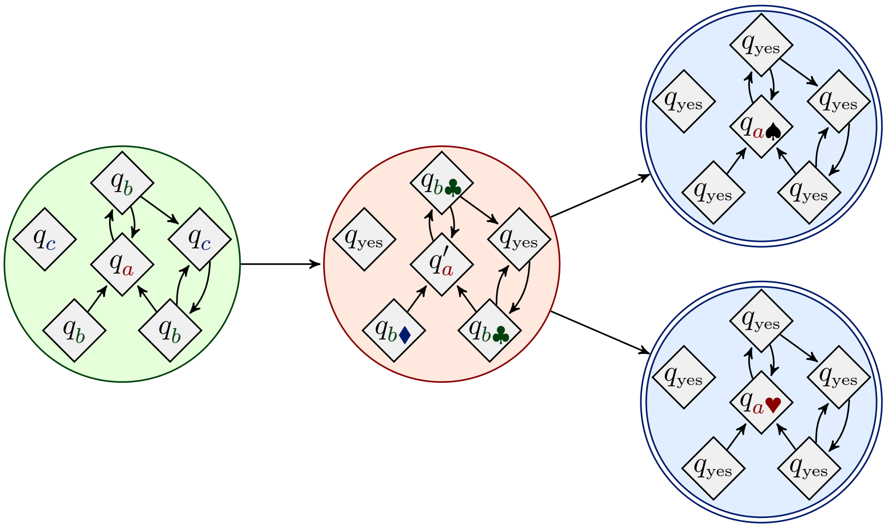


Figure 3.5. An accepting run of the ADGA $\mathcal{A}_{\text{centric}}$ from Fig. 3.3 on the labeled graph $(G_1)_{\lambda_1}$ from Fig. 3.4a.

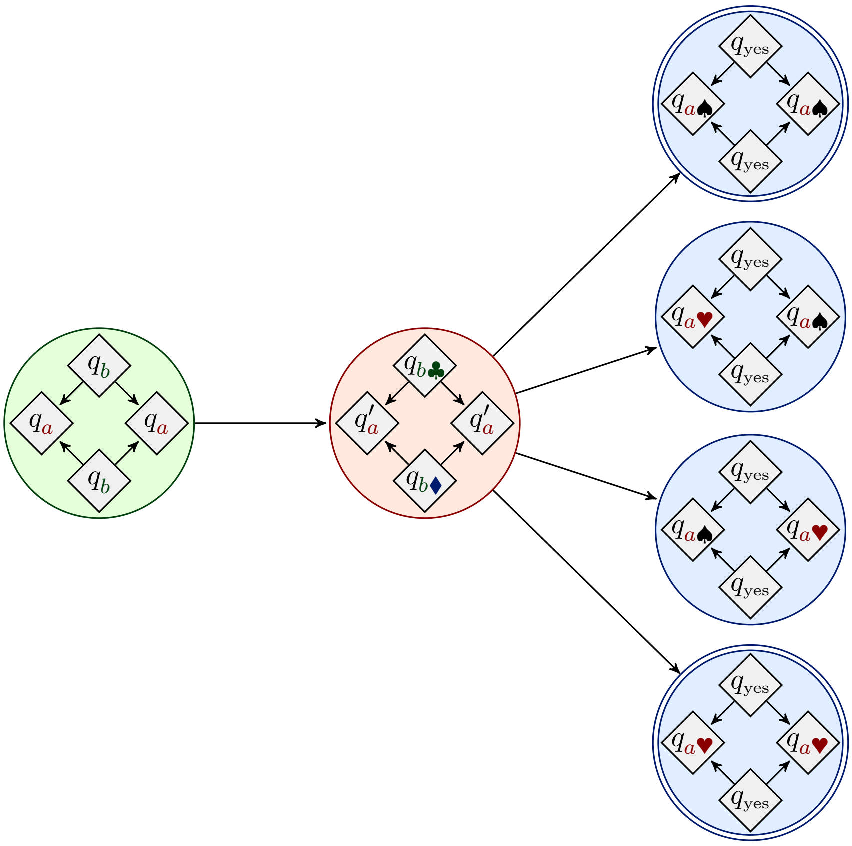


Figure 3.6. A rejecting run of the ADGA $\mathcal{A}_{\text{centric}}$ from Fig. 3.3 on the labeled graph $(G_2)_{\lambda_2}$ from Fig. 3.4b.

while the b- and c-labeled nodes verify that the graph coloring is valid from their point of view. 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 ♣ and ♦. 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 ♣ and ♦ occur, thus ensuring that they have at least two incoming neighbors. Then they simultaneously pick the markers ♠ and ♥, thereby creating different universal branches, and the run of the automaton terminates. Finally, the ADGA checks that all the nodes approve of the graph (meaning that none of them has reached state q_{no}), and that in each universal branch, precisely one of the markers ♠ and ♥ occurs, which implies that there is a unique a-labeled node.

To sum up, the graph language $\mathrm{L}(\mathcal{A}_{\text{centric}})$ consists of all the $\{a, b, c\}$-labeled $\{\square\}$-graphs 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.

An example of a labeled graph that lies in $\mathrm{L}(\mathcal{A}_{\text{centric}})$ is the graph $(G_1)_{\lambda_1}$ from Fig. 3.4a, for which we have seen an accepting run in Fig. 3.5. On the other hand, the labeled graph $(G_2)_{\lambda_2}$ from Fig. 3.4b is not an element of $\mathrm{L}(\mathcal{A}_{\text{centric}})$, since it contains two a-labeled nodes. Either this fact is detected through the universal branching in the second round (as in the run in Fig. 3.6), or the two b-labeled nodes fail to choose two different markers in the first round, leading to refusal by the a-labeled nodes. In any case, the resulting run is rejecting.

3.3 Further Examples

The automaton $\mathcal{A}_{\text{centric}}$ that accompanied us through Section 3.2 has been useful for illustrating several features of ADGAs in a reasonably small example, but its recognized graph language might seem a bit artificial. In this section, we focus on more natural graph properties like being connected, containing a cycle, or being planar, in order to give the reader a better idea of what is possible with ADGAs, and how it can be accomplished.

Connected Graphs

The fact that ADGAs aggregate the states reached in the last round of a run gives them the ability to recognize a particular graph property that a purely distributed algorithm, by its very nature, could not perceive: the property of being connected. For any node alphabet Σ and edge alphabet Γ, we can construct an ADGA $\mathcal{A}_{\ddagger}(\Sigma, \Gamma)$ that recognizes the language of all

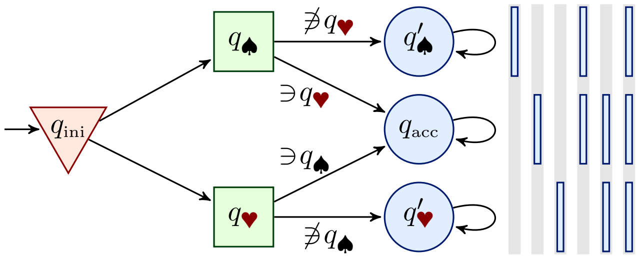


Figure 3.7. $\mathcal{A}_{\ddagger}(\{\square\},\{\square\})$, an ADGA over $\langle\{\square\},\{\square\}\rangle$ whose graph language consists of the (weakly) connected graphs.

(weakly) connected Σ-labeled Γ-graphs. The following example shows this for $\Sigma = \Gamma = \{\square\}$, but other cases are completely analogous.

3.3.1 Example (Recognizing Connected Graphs).
The ADGA $\mathcal{A}_{\ddagger}(\{\square\},\{\square\})$ is specified in Fig. 3.7. It proceeds as follows: In the first round, the nodes simultaneously pick the markers $\spadesuit$ and $\heartsuit$, creating a universal branch for each combination of choices. Then, in the second round, they check in each branch whether all their incoming neighbors have chosen the same marker as themselves. If this is the case, they simply retain their marker. Otherwise, they signalize a discordance in the affected branch by switching to q_{acc}. Afterwards, the automaton accepts the input graph if and only if, in each universal branch, either all the nodes have chosen the same marker, or a discordance has been signalized.

If the input graph is (weakly) connected, and not all nodes have chosen the same marker, then there are always two adjacent nodes with different markers, and at least one of them will signalize a discordance. However, if the graph is not connected, then nodes in different connected components can choose different markers, without any of them detecting it, and hence some branches of the run will reach rejecting configurations (corresponding to $\{q'_\spadesuit, q'_\heartsuit\}$).

In some cases, it is convenient to restrict the allowed input graphs of an automaton to (weakly) connected graphs. To this end, we define the connected graph language of an ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ as

$$\mathrm{L}_{\ddagger}(\mathcal{A}) := \mathrm{L}(\mathcal{A}) \cap \mathrm{L}\big(\mathcal{A}_{\ddagger}(\Sigma, \Gamma)\big).$$

As we will see in Section 3.6, ADGA-recognizable graph languages are effectively closed under intersection, thus we can always replace $\mathcal{A}$ by an intersection automaton of $\mathcal{A}$ and $\mathcal{A}_{\ddagger}(\Sigma, \Gamma)$. Therefore, the definition above is only a convenience.

Directed Trees

One situation in which it is helpful to require connected input graphs is when recognizing the language of directed trees. By a directed tree we mean a connected simple graph, such that there is a unique source, called the *root*, and any other node has exactly one incoming neighbor, called its *parent*.

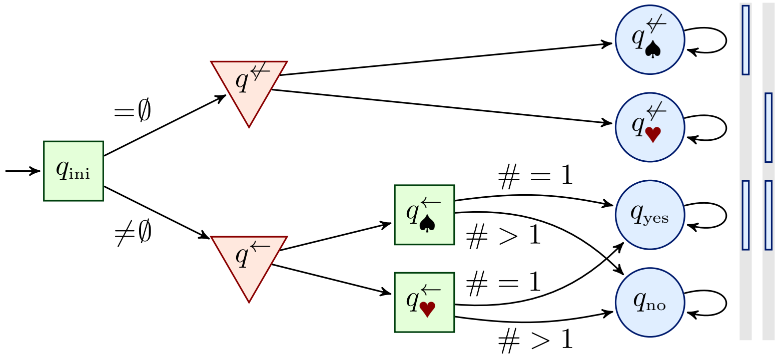


Figure 3.8. $\mathcal{A}_{\mathrm{tree}}$, an ADGA over $\langle\{\square\},\{\square\}\rangle$ whose *connected* graph language $\mathrm{L}_{\ddagger}(\mathcal{A}_{\mathrm{tree}})$ consists of the rooted directed trees with all edges directed away from the root.

3.3.2 Example (Recognizing Directed Trees).
The connected graph language of the ADGA $\mathcal{A}_{\mathrm{tree}}$ specified in Fig. 3.8 is precisely the set of all directed trees. The automaton follows quite naturally from the above definition of a directed tree. First, each node checks whether it has any incoming neighbors. If not, then it expects to be the unique root (indicated by state $q^{\not\leftarrow}$), otherwise it must verify that it has only one parent (indicated by state $q^{\leftarrow}$). In any case, in the second round, each node simultaneously picks the markers ♠ and ♥, which causes a universal branching of the run. Any potential root node terminates at that point. The other nodes continue for another round, and verify in each branch that they see precisely one state in their incoming neighborhood (the symbol "#" in the state diagram refers to the cardinality of the received set). If a node has more than one incoming neighbor, it will see several states in some of the branches, and signalize this error by going to q_{no}. Otherwise it goes to q_{yes}. Finally, the ADGA checks that the last configuration in every branch is error-free and contains only one of the markers ♠ and ♥, thereby ensuring that there is precisely one node that claims to be the root.

Note that the entire approach relies heavily on the requirement that the input graph be connected: without this condition, there could be a directed cycle disconnected from the root.

Undirected Graphs

Up to now, our examples have only involved graphs with directed edges. But, as mentioned in Section 2.2, we can represent undirected graphs as directed graphs with bidirectional edges. The next example shows an ADGA that checks whether a $\{\square\}$-labeled $\{\square\}$-graph is undirected. Again, the principle can be generalized to any node alphabet Σ and edge alphabet Γ, thus giving us an entire family of ADGAs. We denote by $\mathcal{A}_{\upharpoonleft\downharpoonright}(\Sigma,\Gamma)$ the automaton that recognizes the language of all undirected Σ-labeled Γ-graphs.

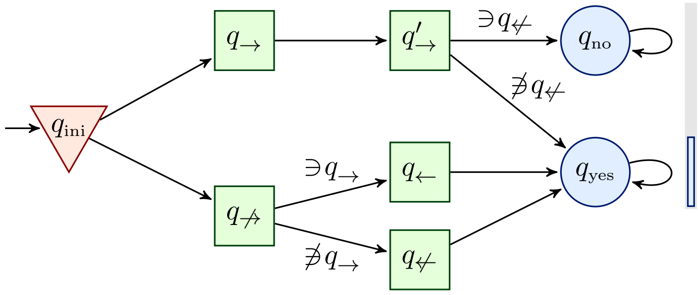


Figure 3.9. $\mathcal{A}_{\upharpoonleft\downharpoonright}(\{\square\},\{\square\})$, an ADGA over $\langle\{\square\},\{\square\}\rangle$ whose graph language consists of the undirected graphs.

3.3.3 Example (Recognizing Undirected Graphs).
The ADGA $\mathcal{A}_{\upharpoonleft\downharpoonright}(\{\square\},\{\square\})$ is specified in Fig. 3.9. It uses a universal branching in the first round, where each node can either send a message to all of its outgoing neighbors (state $q_{\to}$) or remain silent (state $q_{\not\to}$). In the second round, the silent nodes check whether or not they have received any message, which they indicate by going to $q_{\leftarrow}$ or $q_{\not\leftarrow}$, respectively. In the last round, the nodes that have sent a message verify that none of their incoming neighbors report not to have received any message. The automaton then accepts the input graph if and only if every test turns out positive in each universal branch.

Analogously to the connected graph language, we define the undirected graph language of an ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$ as

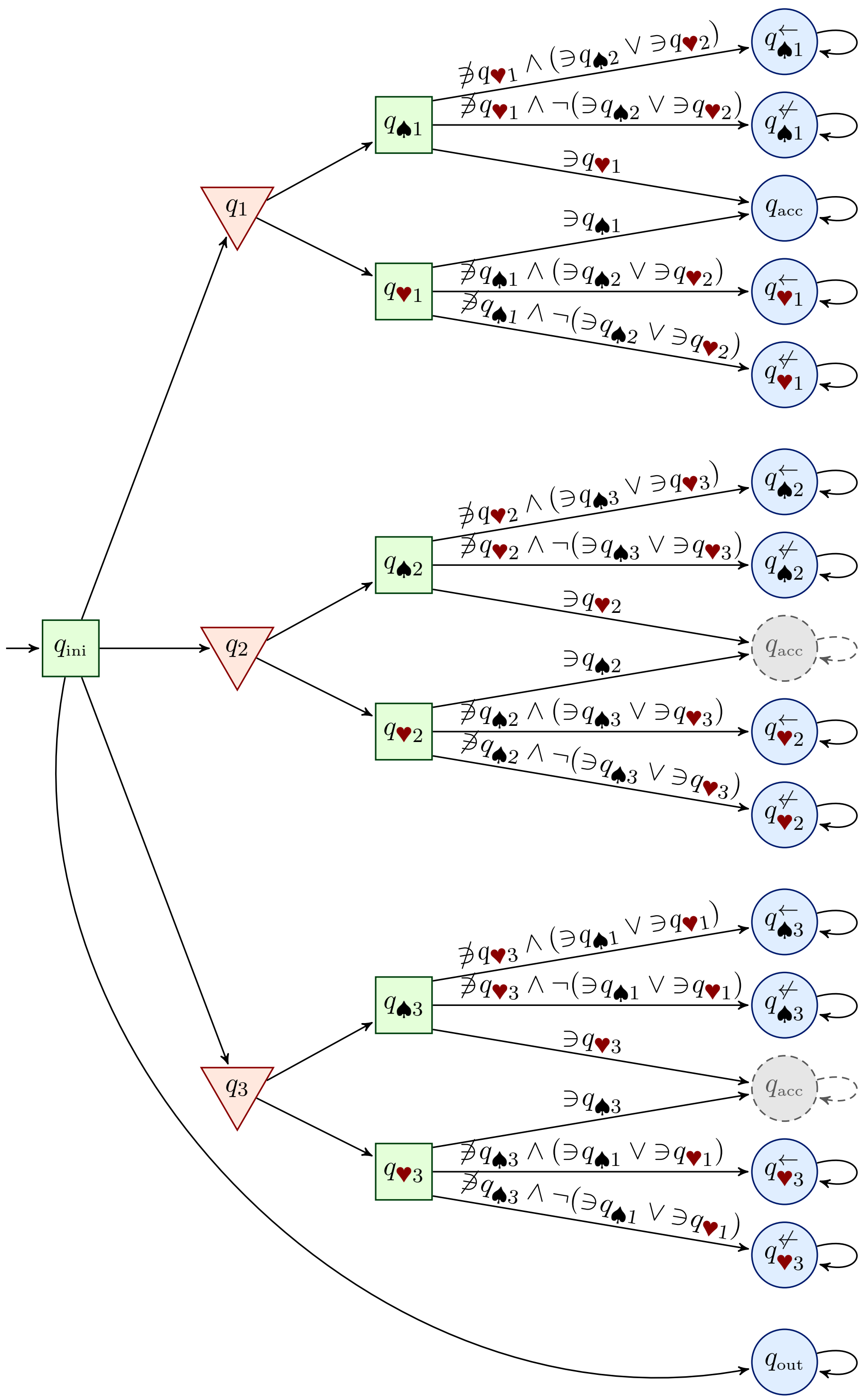
$$\mathrm{L}_{\upharpoonleft\downharpoonright}(\mathcal{A}) \coloneqq \mathrm{L}(\mathcal{A}) \cap \mathrm{L}\big(\mathcal{A}_{\upharpoonleft\downharpoonright}(\Sigma,\Gamma)\big).$$

Graph Minors

To finish our series of examples, we show how ADGAs can check their input graphs for particular minors, and then make use of this to recognize the graphs that contain a cycle and the planar graphs. The approach is heavily inspired by the book [CE12], where it is shown how planarity can be expressed in monadic second-order logic. In this context, we only consider unlabeled, simple, undirected graphs.

For any given loop-free graph H, we can construct an ADGA $\mathcal{A}_H^{\mathrm{minor}}$, such that, for every graph G, the graph language $\mathrm{L}_{\upharpoonleft\downharpoonright}(\mathcal{A}_H^{\mathrm{minor}})$ contains G if and only if H is a minor of G. Our construction follows from the characterization of minor inclusion given in Lemma 2.3.4. The automaton proceeds as follows: First, it nondeterministically partitions some subset of V_G into sets $U_1,\dots,U_n$, corresponding to the n nodes of H. Then, it checks that each induced subgraph $G[U_i]$ is connected, and that for each edge in H between two nodes v_i and v_j, there is an edge in G connecting the corresponding subgraphs $G[U_i]$ and $G[U_j]$. In order to verify that a subgraph is connected, we use a slightly adapted version of the automaton $\mathcal{A}_{\ddagger}(\{\square\},\{\square\})$ from Fig. 3.7 as a building block.

The following example shows the construction for the complete graph K_3, which by Remark 2.3.3 gives us an ADGA for the language of all simple undirected graphs that contain at least one cycle of three or more nodes.



$$\ni q_{\text{acc}} \vee \bigwedge_{1\le j\le 3}\Big(\big(\ni q^{\leftarrow}_{\spadesuit j} \wedge \not\ni q^{\leftarrow}_{\heartsuit j} \wedge \not\ni q^{\not\leftarrow}_{\heartsuit j}\big) \vee \big(\ni q^{\leftarrow}_{\heartsuit j} \wedge \not\ni q^{\leftarrow}_{\spadesuit j} \wedge \not\ni q^{\not\leftarrow}_{\spadesuit j}\big)\Big)$$

Figure 3.10. $\mathcal{A}^{\text{minor}}_{\mathrm{K}_3}$, an ADGA over $\langle\{\Box\},\{\Box\}\rangle$ whose *undirected* graph language $\mathrm{L}_{\upharpoonleft\downharpoonright}(\mathcal{A}^{\text{minor}}_{\mathrm{K}_3})$ consists of the graphs that contain K_3 as a minor, or in other words, the graphs that contain at least one cycle of three or more nodes.

3.3.4 Example (Recognizing Graphs with a Cycle).
The ADGA $\mathcal{A}^{\mathrm{minor}}_{\mathrm{K}_3}$ is specified in Fig. 3.10. Since it has too many accepting sets to represent them with a barcode, its acceptance condition is given as a set formula. Also, for the sake of better readability, there are three occurrences of q_{acc} in the state diagram, but they all represent the same state.

Let U_1, U_2 and U_3 be three sets corresponding to the three nodes of K_3. In the first round, each node of the input graph nondeterministically decides whether to participate and join one of those sets (states q_1, q_2, q_3), or not to interfere at all and terminate right away (state q_{out}). Then, in the second round, there is a universal branching in which each participating node simultaneously picks the markers ♠ and ♥. In the third and last round, in every universal branch, each participating node checks whether it has any neighbor that is in the same set as itself but has chosen a different marker. If this is the case, it goes to q_{acc} to signalize that the affected branch of the run contains a discordance and must be treated accordingly by the automaton. (This is analogous to the behaviour of the ADGA from Example 3.3.1.) Otherwise, the node assumes that the subgraph $G[U_i]$ to which it belongs is connected, and it checks whether any of its neighbors is part of the subgraph $G[U_{(i \bmod 3)+1}]$. If so, it switches to a state with a superscript "$\leftarrow$", otherwise to a state with a "$\not\leftarrow$". Finally, the automaton decides on acceptance as follows: Branches in which a discordance has been signalized (by the state q_{acc}) are inconclusive, and thus the configurations reached in such branches are considered to be accepting. In all the other branches, the automaton expects that in each subgraph $G[U_i]$, the nodes agree on one of the markers ♠ and ♥, and at least one of them signalizes that it is connected to a node in the "next" subgraph $G[U_{(i \bmod 3)+1}]$.

We can proceed similarly to construct the automaton $\mathcal{A}^{\mathrm{minor}}_{H}$ for any other loop-free graph H. Levels 0, 1 and 2 are completely analogous and depend only on the number of nodes of H. Level 4, on the other hand, must be adapted to the structure of the graph. For each edge in H between two nodes v_i and v_j, it must be verified that there is a corresponding edge in the input graph G, connecting $G[U_i]$ and $G[U_j]$. Either the nodes in U_i or the nodes in U_j must perform this verification. However, the number of permanent states required by the nodes in each set U_i grows exponentially with the number of edges for which those nodes are responsible. This is because a single node in U_i might have neighbors in several other sets, and each combination must be encoded in a separate state. Hence, if we want to keep the total number of states low, we have to balance the load among the sets as evenly as possible. The exact specification of $\mathcal{A}^{\mathrm{minor}}_{H}$ can thus be optimized for each graph H, but the construction principle is always the same as for $\mathcal{A}^{\mathrm{minor}}_{\mathrm{K}_3}$.

The possibility to check for arbitrary minors is a powerful tool. As another application example, we outline how to recognize the language of planar graphs.

3.3.5 Example (Recognizing Planar Graphs).
By the Kuratowski-Wagner Theorem (Theorem 2.3.5), a graph is planar if and only if it contains neither the complete graph K_5 nor the complete bipartite graph $\mathrm{K}_{3,3}$ as a minor. Moreover, as we will see in Section 3.6, ADGA-recognizable graph languages are effectively closed under boolean set operations.

Thus, by constructing the union automaton of $\mathcal{A}^{\mathrm{minor}}_{\mathrm{K}_5}$ and $\mathcal{A}^{\mathrm{minor}}_{\mathrm{K}_{3,3}}$ and then complementing it, we obtain an ADGA $\mathcal{A}_{\mathrm{planar}}$ over $\langle\{\square\},\{\square\}\rangle$ whose undirected graph language is precisely the set of all simple undirected planar graphs.

3.4 Normal Forms

In this section, we establish some normal forms of ADGAs, which will prove helpful for the closure constructions in Section 3.6.

The notion of a nonblocking ADGA is analogous to that of a nonblocking finite automaton on words: it guarantees that the automaton cannot "get stuck" during execution, which for an ADGA means that all of its runs eventually reach a permanent configuration in each universal branch.

3.4.1 Definition (Nonblocking ADGA).
An ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$ is called nonblocking if and only if every configuration $G_\kappa \in Q^{(\Gamma)}$ that is reachable by $\mathcal{A}$ has at least one successor configuration, i.e., $\delta^{✿}(G_\kappa) \neq \emptyset$.

A sufficient (but not necessary) condition for $\mathcal{A}$ to be nonblocking is that its transition function is complete, i.e., $\delta(q,\widehat{S}) \neq \emptyset$ for every $q \in Q$ and $\widehat{S} \in (2^Q)^\Gamma$. This gives us an effective way of transforming any given ADGA into an equivalent nonblocking one.

3.4.2 Remark.
For every ADGA $\mathcal{A}$, we can effectively construct an equivalent ADGA $\mathcal{A}'$ that is nonblocking. Moreover, $\mathrm{siz}(\mathcal{A}') \leq \mathrm{siz}(\mathcal{A}) + \mathrm{len}(\mathcal{A})$ and $\mathrm{len}(\mathcal{A}') = \mathrm{len}(\mathcal{A})$.

Proof. Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$. We extend $\mathcal{A}$ such that its transition function becomes complete, giving us an equivalent ADGA that is guaranteed to be nonblocking. To this end, we introduce an additional permanent state q_i^{stop} for every nonpermanent level i. If a node was blocked at level i, it now simply moves to state q_i^{stop}, and waits there for the other nodes to terminate. A permanent configuration is then accepting if it already was so previously, or if it contains states indicating that the lowest level at which some node would have been blocked, is universal. Formally, we fix the set $Q_{\mathrm{stop}} = \{q_i^{\mathrm{stop}} \mid 0 \leq i < \mathrm{len}(\mathcal{A})\}$, and construct $\mathcal{A}' = \langle \Sigma, \Gamma, \widehat{Q}', \sigma, \delta', \mathcal{F}'\rangle$, with

- $Q'_\exists = Q_\exists, \quad Q'_\forall = Q_\forall, \quad Q'_\mathrm{P} = Q_\mathrm{P} \cup Q_{\mathrm{stop}},$
- $\delta'(q,\widehat{S}) = \begin{cases} \delta(q,\widehat{S}) & \text{if } \widehat{S} \in (2^Q)^\Gamma \text{ and } \delta(q,\widehat{S}) \neq \emptyset, \\ \{q^{\mathrm{stop}}_{\mathrm{l}_\mathcal{A}(q)}\} & \text{otherwise,} \end{cases}$
  for every $q \in Q'_\mathrm{N}$ and $\widehat{S} \in (2^{Q'})^\Gamma$, and
- $\mathcal{F}' = \mathcal{F} \cup \bigl\{F \subseteq Q'_P \bigm| \min\{i \mid q_i^{\mathrm{stop}} \in F\} \text{ is a universal level of } \mathcal{A}\bigr\}.$

As a slight optimization, if $Q_\exists = \emptyset$ or $Q_\forall = \emptyset$, the states in Q_{stop} can be merged into a single state q_{stop}. ■

Next, again in analogy to finite automata on words, we say that an ADGA is trim if it does not have any states that are obviously useless. In the case of (nondeterministic) finite automata on words, a state is considered useless if it is not reachable from any initial state or if no accepting state is reachable from it. However, for ADGAs the notion of reachability of states is more involved, since it is subject to the reachability of configurations. For our purposes, it will be enough to consider a necessary condition for reachability, which can be easily checked for every state: reachability within the state diagram, ignoring the requirements on the transition arrows. States that do not satisfy this condition are obviously useless, since they cannot occur in any run. We can thus safely remove them from the automaton, without affecting the graph language it recognizes.

3.4.3 Definition (Trim ADGA).
Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be an ADGA. We consider a state $q \in Q$ to be potentially reachable if

- $q = \sigma(a)$ for some $a \in \Sigma$, or
- $q \in \delta(p, \widehat{S})$ for some $p \in Q$ and $\widehat{S} = \langle S_\gamma \rangle_{\gamma \in \Gamma} \in (2^Q)^\Gamma$, such that p and every state $p' \in \bigcup_{\gamma \in \Gamma} S_\gamma$ are potentially reachable.

The automaton $\mathcal{A}$ is said to be trim if all of its states are potentially reachable.

3.4.4 Remark.
For every ADGA $\mathcal{A}$, we obtain an equivalent ADGA $\mathcal{A}'$ that is trim, by removing all states from $\mathcal{A}$ that are not potentially reachable (and adapting the transition function and acceptance condition accordingly). Moreover, if $\mathcal{A}$ is nonblocking, then so is $\mathcal{A}'$.

Last, we introduce the notion of alternating normal form, which requires that successive nonpermanent levels of an ADGA are alternately existential and universal. Every ADGA can be transformed into an equivalent automaton in alternating normal form by inserting "dummy" levels between any two consecutive levels that are of the same type.

3.4.5 Definition (Alternating Normal Form).
An ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ is in alternating normal form if for every level $i \in \{0, \dots, \operatorname{len}(\mathcal{A}) - 2\}$,

$$\begin{aligned} \operatorname{lev}_i(\mathcal{A}) \subseteq Q_\exists \quad &\text{implies} \quad \operatorname{lev}_{i+1}(\mathcal{A}) \subseteq Q_\forall, \quad \text{and} \\ \operatorname{lev}_i(\mathcal{A}) \subseteq Q_\forall \quad &\text{implies} \quad \operatorname{lev}_{i+1}(\mathcal{A}) \subseteq Q_\exists. \end{aligned}$$

3.4.6 Remark.
For every ADGA $\mathcal{A}$, we can effectively construct an equivalent ADGA $\mathcal{A}'$ that is in alternating normal form. If $\mathcal{A}$ is nonblocking or trim, then these properties carry over to $\mathcal{A}'$. Moreover, $\operatorname{siz}(\mathcal{A}') < 2\operatorname{siz}(\mathcal{A})$ and $\operatorname{len}(\mathcal{A}') < 2\operatorname{len}(\mathcal{A})$.

Proof. Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$. We construct $\mathcal{A}' = \langle \Sigma, \Gamma, \widehat{Q}', \sigma, \delta', \mathcal{F} \rangle$ as an extended version of $\mathcal{A}$, with $Q_\exists \subseteq Q'_\exists$, $Q_\forall \subseteq Q'_\forall$, and $Q_\mathrm{P} = Q'_\mathrm{P}$.

Suppose that there is some level i, with $0 \le i \le \operatorname{len}(\mathcal{A}) - 2$, such that both i and $i+1$ are existential levels in $\mathcal{A}$. We remedy this in $\mathcal{A}'$ by inserting a disjoint copy Q'_{i+1} of $\operatorname{lev}_{i+1}(\mathcal{A})$ "between" $\operatorname{lev}_i(\mathcal{A})$ and $\operatorname{lev}_{i+1}(\mathcal{A})$, and defining the states in Q'_{i+1} to be universal, i.e., $Q'_{i+1} \subseteq Q'_{\forall}$. Then, we redirect the outgoing transitions of states in $\mathrm{lev}_i(\mathcal{A})$, such that, instead of going to states in $\mathrm{lev}_{i+1}(\mathcal{A})$, they go to the corresponding copies in Q'_{i+1}. Turning to these copies, we direct all their outgoing transitions to the matching original states in $\mathrm{lev}_{i+1}(\mathcal{A})$.

More formally, for every state $q' \in Q'_{i+1}$, we denote by q the state in $\mathrm{lev}_{i+1}(\mathcal{A})$ to which it corresponds (i.e., q' is the copy of q). Now, for every $p \in \mathrm{lev}_i(\mathcal{A})$ and $\widehat{S} \in (2^Q)^\Gamma$, we define

$$\delta'(p, \widehat{S}) = \left\{ q' \in Q'_{i+1} \;\middle|\; q \in \delta(p, \widehat{S}) \right\} \cup \left(\delta(p, \widehat{S}) \cap Q_{\mathrm{P}} \right),$$

and for every $q' \in Q'_{i+1}$ and $\widehat{S} \in (2^{Q'})^\Gamma$, we set

$$\delta'(q', \widehat{S}) = \{q\}.$$

In the dual case, where the levels i and $i+1$ are both universal in $\mathcal{A}$, we proceed analogously. By doing so for every level in $\mathcal{A}$ that is directly followed by a level of the same type, and otherwise retaining the original transitions of $\mathcal{A}$, we achieve that $\mathcal{A}'$ is in alternating normal form. Each level $i \in \{1, \dots, \mathrm{len}(\mathcal{A}) - 1\}$ is duplicated at most once, thus $\mathcal{A}'$ cannot exceed twice the size and length of $\mathcal{A}$.

Since the additional levels that we have introduced merely cause the runs of $\mathcal{A}'$ to be longer than those of $\mathcal{A}$, without affecting which permanent configurations are eventually reached, $\mathcal{A}'$ is obviously equivalent to $\mathcal{A}$. It is also easy to see that if $\mathcal{A}$ is trim, so is $\mathcal{A}'$. Finally, we observe that a configuration can only be reachable by $\mathcal{A}'$ if it is also reachable by $\mathcal{A}$, or if it comprises only states from $Q' \setminus Q$ (i.e., states from the additional levels that we have introduced). By construction, any reachable configuration of $\mathcal{A}$ has as many successor configurations in $\mathcal{A}$ as in $\mathcal{A}'$. Furthermore, every configuration comprising only states from $Q' \setminus Q$ has precisely one successor configuration in $\mathcal{A}'$. Hence, if $\mathcal{A}$ is nonblocking, so is $\mathcal{A}'$. ∎

3.5 Game-Theoretic Characterization

In this section, we give an alternative characterization of the acceptance behaviour of ADGAs, using a game-theoretic approach. This different point of view will be useful on two occasions: in Section 3.6, where we will show that ADGAs can be easily complemented, and in Section 4.2, where we will encode the behaviour of a given ADGA into a logical formula.

The entire approach is heavily inspired by the work of Löding and Thomas in [LT00], where they investigated the complementation of finite automata on infinite words. A simplified variant for automata on finite words can be found in [Kum06].

We consider games with two players: the *automaton* (player $\exists$), and the *pathfinder* (player $\forall$).(2) Given an ADGA $\mathcal{A}$ and a labeled graph G_λ, the goal of the automaton is to accept G_λ, while the pathfinder tries to reject

(2) The custom of calling the players "automaton" and "pathfinder" was introduced by Gurevich and Harrington in [GH82].

it. In a way, the automaton wants to come up with an accepting run of $\mathcal{A}$ on G_λ, and the pathfinder seeks to refute any possible run, by finding (a path to) a rejecting configuration. Thus, the automaton is responsible for the nondeterministic (existential) choices, whereas the pathfinder picks among the universal branches.

The game associated with $\mathcal{A}$ and G_λ is represented by a directed acyclic graph whose nodes are configurations of $\mathcal{A}$ on G. This graph can be thought of as a superposition of all possible runs of $\mathcal{A}$ on G_λ. We refer to its nodes as (game) positions, but keep the terminology used for configurations (e.g., "existential", "permanent", etc.). The nonpermanent positions are divided among the two players: existential ones belong to the automaton, universal ones to the pathfinder. Starting at the initial configuration of $\mathcal{A}$ on G_λ, the two players move through the graph together. At each position, the player owning that position has to choose the next move along one of the outgoing edges. This continues until some permanent position is reached. If that position is accepting, the automaton wins, otherwise the pathfinder wins. Also, if a nonpermanent position is reached from which no move is possible, the owner of that position loses. (This cannot happen if $\mathcal{A}$ is nonblocking.)

3.5.1 Definition (Game).
Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be an ADGA and G_λ a Σ-labeled Γ-graph. The game $\mathrm{J}(\mathcal{A}, G_\lambda)$ associated with $\mathcal{A}$ and G_λ is the tuple $\langle \widehat{K}, G_{\kappa_0}, \rightharpoonup \rangle$ defined as follows:

- $\widehat{K} = \langle K_\exists, K_\forall, K_{\mathrm{acc}}, K_{\mathrm{rej}} \rangle$, where
  - $K_\exists \subseteq \langle Q_\exists, Q_\mathrm{P} \rangle^G$ and $K_\forall \subseteq \langle Q_\forall, Q_\mathrm{P} \rangle^G$ are the sets of existential and universal configurations, respectively, reachable by $\mathcal{A}$ on G_λ,
  - $K_{\mathrm{acc}}, K_{\mathrm{rej}} \subseteq Q_\mathrm{P}^G$ are the sets of accepting and rejecting configurations, respectively, reachable by $\mathcal{A}$ on G_λ.

  We use the abbreviation $K \coloneqq K_\exists \cup K_\forall \cup K_{\mathrm{acc}} \cup K_{\mathrm{rej}}$.
- $G_{\kappa_0} = G_{\sigma \circ \lambda}$ is the starting position of the game.
- $\rightharpoonup \subseteq K \times K$ is the set of directed edges for which $\langle K, \rightharpoonup \rangle$ constitutes a directed acyclic graph, such that
  - G_{κ_0} is the only source,
  - every node $G_\kappa \in (K_\exists \cup K_\forall)$ with $\delta^{✿}(G_\kappa) = \{G_{\mu_1}, \ldots, G_{\mu_m}\}$ has exactly m outgoing neighbors $G_{\mu_1}, \ldots, G_{\mu_m} \in K$, and
  - every node $G_\kappa \in (K_{\mathrm{acc}} \cup K_{\mathrm{rej}})$ is a sink.

If $\mathcal{A}$ and G_λ are not relevant in a given context, we refer to $\mathrm{J}(\mathcal{A}, G_\lambda)$ simply as *a* game $J = \langle \widehat{K}, G_{\kappa_0}, \rightharpoonup \rangle$. For convenience, we will apply graph-theoretic notions directly to J, referring implicitly to its underlying graph $\langle K, \rightharpoonup \rangle$, e.g., by "a path in J" we mean "a path in $\langle K, \rightharpoonup \rangle$".

3.5.2 Example (Game associated with $\mathcal{A}_{\text{centric}}$).
Figure 3.11 represents the game $\mathrm{J}\big(\mathcal{A}_{\text{centric}}, (G_2)_{\lambda_2}\big)$ associated with the ADGA from Fig. 3.3 and the labeled graph from Fig. 3.4b. The green configuration is the starting position and belongs to the automaton, whereas the red positions belong to the pathfinder. Just as for runs, the blue positions with a double circle are accepting, and the other blue positions rejecting.

The game is a superposition of all possible runs of $\mathcal{A}_{\text{centric}}$ on $(G_2)_{\lambda_2}$, in the sense that it contains all of these runs as subgraphs. For instance, we obtain the run from Fig. 3.6 as the subgraph induced by the green position, the third red position from the top, and the four bottommost blue positions.

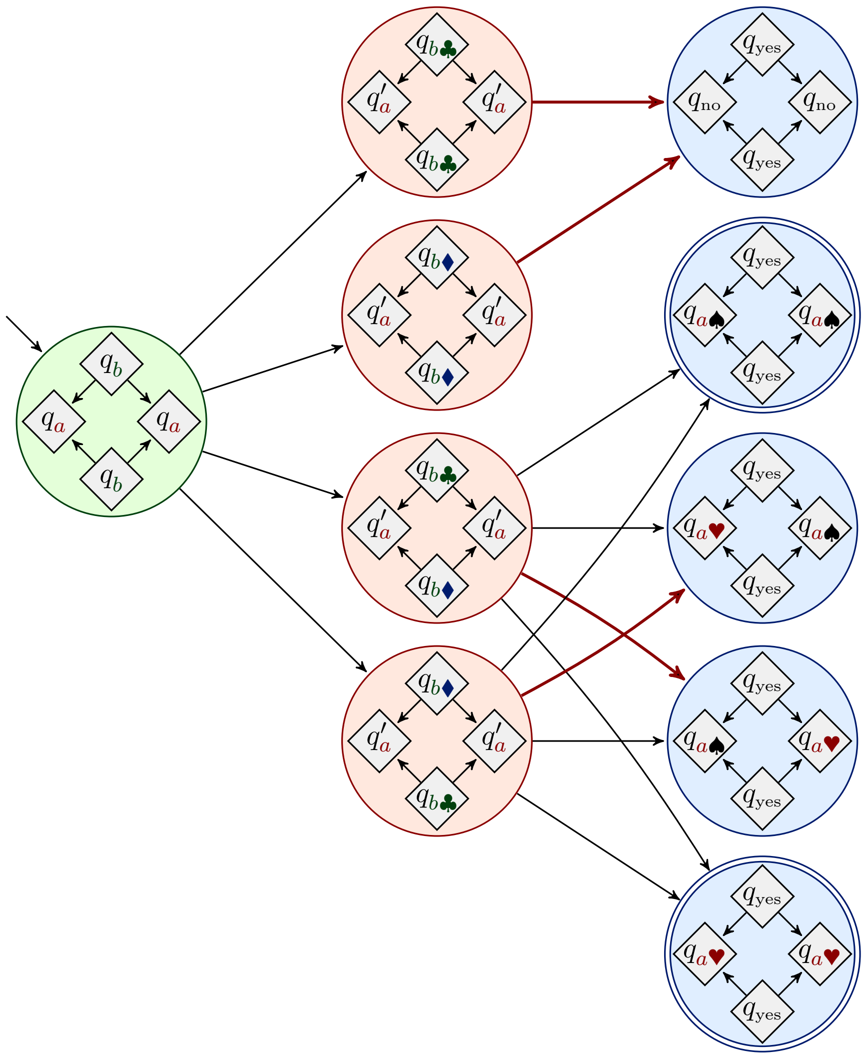


Figure 3.11. The game associated with the ADGA $\mathcal{A}_{\text{centric}}$ from Fig. 3.3 and the labeled graph $(G_2)_{\lambda_2}$ from Fig. 3.4b. The edges highlighted in red represent a winning strategy for the pathfinder.

It remains to formalize how a game is played, and how the winner is determined.

3.5.3 Definition (Play).
A play π in a game J is a path from the starting position to some sink (i.e., a maximal path). The winner of the play $\pi = G_{\kappa_0} \cdots G_{\kappa_n}$ (with respect to J) is

- the automaton, if G_{κ_n} is either accepting or universal,
- the pathfinder, if G_{κ_n} is either rejecting or existential.

We will refer to G_{κ_n} as a goal position of the automaton or the pathfinder, depending on which player wins when reaching that position.

The moves chosen by the two players in a play π are determined by their respective strategies.

3.5.4 Definition (Strategy).
A (memoryless or positional) strategy for player $X \in \{\exists, \forall\}$ in a game $J = \langle \widehat{K}, G_{\kappa_0}, \rightharpoonup \rangle$ is a partial function $f_X \colon K_X \to K$, such that $f_X(G_\kappa)$ is an outgoing neighbor of G_κ, for every position G_κ in J that belongs to player X and is not a sink. A play $\pi = G_{\kappa_0} \cdots G_{\kappa_n}$ is played according to f_X if and only if $f_X(G_{\kappa_i}) = G_{\kappa_{i+1}}$ for every node G_{κ_i} on π that belongs to player X, where $0 \le i < n$.

We say that f_X is a winning strategy for player X if and only if that player wins every play π in J played according to f_X.

3.5.5 Example (Winning Strategy for the Pathfinder).
We consider again the game $\mathrm{J}\big(\mathcal{A}_{\text{centric}}, (G_2)_{\lambda_2}\big)$ from Fig. 3.11. No matter which position the automaton chooses in the first round, the pathfinder can always move to a rejecting position in the second round. One possible winning strategy for the pathfinder is represented by the edges highlighted in red.

The fact that the pathfinder has a winning strategy in this game is essentially a restatement of the observation made in Example 3.2.8: every run of $\mathcal{A}_{\text{centric}}$ on $(G_2)_{\lambda_2}$ is rejecting.

The previous example already suggests a strong relationship between the acceptance behaviour of ADGAs and the winning strategies of the two players. To no great surprise, both concepts turn out to be equivalent.

3.5.6 Lemma (Acceptance and Winning Strategy).
Let $\mathcal{A}$ be an ADGA and G_λ a labeled graph. Then $\mathcal{A}$ accepts G_λ if and only if the automaton has a winning strategy in the game $\mathrm{J}(\mathcal{A}, G_\lambda)$.

Proof. We give a very simple proof, for the sake of completeness.

($\Rightarrow$) If $\mathcal{A}$ accepts G_λ, there is an accepting run R of $\mathcal{A}$ on G_λ, from which we construct the following strategy $f_\exists$ for the automaton in $J = \mathrm{J}(\mathcal{A}, G_\lambda)$: For every position G_κ in J that belongs to the automaton and is not a sink, $f_\exists(G_\kappa)$ is the unique outgoing neighbor of G_κ in R, provided that G_κ occurs in R. Otherwise $f_\exists(G_\kappa)$ is some arbitrary outgoing neighbor of G_κ in J. Hence, any play $\pi = G_{\sigma\circ\lambda} \cdots G_\kappa$ in J played according to $f_\exists$ corresponds to a maximal path in R (starting at $G_{\sigma\circ\lambda}$). Since R is a run and G_κ has no outgoing neighbors, G_κ is either universal or permanent.

Furthermore, since R is accepting, if G_κ is permanent, then it is also accepting. In any case, the automaton wins the play π, i.e., it wins every play in J played according to $f_\exists$, which means that $f_\exists$ is a winning strategy for that player.

($\Leftarrow$) If the automaton has a winning strategy $f_\exists$ in $J = \mathrm{J}(\mathcal{A}, G_\lambda)$, we can construct the following run R of $\mathcal{A}$ on G_λ: Starting at $G_{\sigma\circ\lambda}$, for every node G_κ in R, if G_κ belongs to the automaton, then its unique outgoing neighbor in R is $f_\exists(G_\kappa)$ [(3)], and if G_κ belongs to the pathfinder, then its outgoing neighbors in R are given by $\delta^{✪}(G_\kappa)$. Hence, every maximal path $\pi = G_{\sigma\circ\lambda} \cdots G_\kappa$ in R corresponds to a play in J played according to $f_\exists$. Since permanent configurations do not have any outgoing neighbors, the only node on π that might be permanent is G_κ. Furthermore, since $f_\exists$ is a winning strategy for the automaton, the configuration G_κ is either accepting or universal. Thus every permanent configuration occurring on some (maximal) path in R is accepting, which implies that R is an accepting run, from which follows that $\mathcal{A}$ accepts G_λ. ■

Dually to Lemma 3.5.6, an ADGA rejects a labeled graph if and only if the pathfinder has a winning strategy in the associated game. Instead of proving this directly, we can infer it from the following determinacy result.

3.5.7 Lemma (Determinacy).
In every game J, either the automaton or the pathfinder has a winning strategy.

Proof. We consider every position G_κ of the game $J = \langle \widehat{K}, G_{\kappa_0}, \rightharpoonup \rangle$ as the starting position of a subgame J_κ which is obtained by restricting J to the subgraph induced by G_κ and all its descendant configurations. Note that this implies that $J_{\kappa_0} = J$. We show by induction on the structure of the game, that for every position G_κ in J, either the automaton or the pathfinder has a winning strategy in the induced subgame J_κ.

(BC) If G_κ is a sink, then J_κ consists only of the single position G_κ, and the only possible play in J_κ is $\pi = G_\kappa$. The automaton wins that play if G_κ is either accepting or universal, otherwise the pathfinder wins. In any case, one of the two players has a (trivial) winning strategy.

(IS) Now consider the case that G_κ has $m > 0$ outgoing neighbors $G_{\mu_1}, \dots, G_{\mu_m}$. Let player $X \in \{\exists, \forall\}$ be the player who has to make a move at position G_κ, i.e., $G_\kappa \in K_X$, and let player Y be the opponent. By induction hypothesis, we know that for each of the subgames $J_{\mu_1}, \dots, J_{\mu_m}$, either player X or player Y has a winning strategy. There are two possible cases:

- If player X has a winning strategy f_X in some subgame J_{μ_i} ($1 \leq i \leq m$), this strategy can be extended to a winning strategy f'_X in

[(3)] The value of $f_\exists(G_\kappa)$ cannot be undefined, because otherwise any path in R from $G_{\sigma\circ\lambda}$ to G_κ would be a play played according to $f_\exists$ that is lost by the automaton (since G_κ is existential), which would contradict the assumption that $f_\exists$ is a winning strategy for the automaton.

J_κ, where for every node G_ν in J_κ that belongs to player X and has at least one outgoing neighbor,

$$f'_X(G_\nu) = \begin{cases} G_{\mu_i} & \text{if } G_\nu = G_\kappa, \\ f_X(G_\nu) & \text{if } G_\nu \text{ is in } J_{\mu_i}, \\ G_{\nu'} & \text{otherwise, where } G_{\nu'} \text{ is some arbitrary} \\ & \text{outgoing neighbor of } G_\nu \text{ in } J_\kappa. \end{cases}$$

– Otherwise, player Y has winning strategies $f_{Y1}, \dots, f_{Ym}$ for each of the subgames $J_{\mu_1}, \dots, J_{\mu_m}$, respectively. These can be combined into a winning strategy f'_Y in J_κ, such that for every node G_ν in J_κ that belongs to player Y and has at least one outgoing neighbor, $f'_Y(G_\nu) = f_{Yi}(G_\nu)$, where $i \in \{1, \dots, m\}$ is the smallest[(4)] index for which the corresponding subgame J_{μ_i} contains G_ν.

In both cases, either the automaton or the pathfinder has a winning strategy in J_κ. ■

3.6 Closure Properties

Building on the results from Sections 3.4 and 3.5, we can now establish some closure properties of the class of ADGA-recognizable graph languages.

Complementation can be achieved by a simple dualization construction, which does not involve any blow-up. We have already used this implicitly in the examples of Section 3.1, where the ADGA $\mathcal{A}_3^{\text{color}}$ from Fig. 3.1 was complemented by changing its existential states to universal ones and complementing its acceptance condition. This resulted in the ADGA $\overline{\mathcal{A}}_3^{\text{color}}$ from Fig. 3.2. The following definition generalizes this construction for arbitrary ADGAs.

3.6.1 Definition (Dual Automaton).
Let $\mathcal{A} = \langle \Sigma, \Gamma, \langle Q_\exists, Q_\forall, Q_\mathrm{P} \rangle, \sigma, \delta, \mathcal{F} \rangle$ be an ADGA. Its dual automaton $\overline{\mathcal{A}}$ is obtained by swapping the existential and universal states, and complementing the set of accepting states, i.e.,

$$\overline{\mathcal{A}} = \langle \Sigma, \Gamma, \langle Q_\forall, Q_\exists, Q_\mathrm{P} \rangle, \sigma, \delta, 2^{Q_\mathrm{P}} \setminus \mathcal{F} \rangle.$$

To show that the dual automaton is always a complement automaton, we first look at this construction from the game-theoretic point of view.

3.6.2 Lemma.
Consider an ADGA $\mathcal{A}$ over $\langle \Sigma, \Gamma \rangle$ and a labeled graph $G_\lambda \in \Sigma^{\textcircled{\scriptsize$\Gamma$}}$. Then the automaton has a winning strategy in the game $\mathrm{J}(\mathcal{A}, G_\lambda)$ if and only if the pathfinder has a winning strategy in the dual game $\mathrm{J}(\overline{\mathcal{A}}, G_\lambda)$.

Proof. Let $J = \mathrm{J}(\mathcal{A}, G_\lambda) = \langle \langle K_\exists, K_\forall, K_\mathrm{acc}, K_\mathrm{rej} \rangle, G_{\kappa_0}, \rightharpoonup \rangle$. We observe that the dual game $\overline{J} = \mathrm{J}(\overline{\mathcal{A}}, G_\lambda)$ has the same underlying graph and starting

(4) The subgames $J_{\mu_1}, \dots, J_{\mu_m}$ are not necessarily disjoint. If a position occurs in several subgames, player Y can arbitrarily choose which winning strategy to follow at that position, since any choice will lead the play one step closer to some goal position of player Y.

position as J, only the roles and winning conditions of the two players have been interchanged, i.e., $\bar{J} = \langle \langle K_{\forall}, K_{\exists}, K_{\mathrm{rej}}, K_{\mathrm{acc}} \rangle, G_{\kappa_0}, \rightharpoonup \rangle$. Hence, every play π in J is also a play in $\bar{J}$, and vice versa. Due to the complementarity of the winning conditions, player $X \in \{\exists, \forall\}$ wins π in J if and only if its opponent, player Y, wins π in $\bar{J}$. Moreover, the reversal of roles ensures that a strategy f for player X in the one game is a strategy for player Y in the other game. Thus, player X wins every play played according f in J if and only if player Y wins every play played according f in $\bar{J}$. ■

It is now straightforward to prove the desired result.

3.6.3 Lemma (Complementation).
For every ADGA $\mathcal{A}$ over $\langle \Sigma, \Gamma \rangle$, the dual automaton $\bar{\mathcal{A}}$ recognizes the complement language of $\mathcal{A}$, i.e.,

$$\mathrm{L}(\bar{\mathcal{A}}) = \Sigma^{\circledR} \setminus \mathrm{L}(\mathcal{A}).$$

Proof. Let $G_\lambda \in \Sigma^{\circledR}$. By Lemma 3.5.6, $\mathcal{A}$ accepts G_λ if and only if the automaton has a winning strategy in the game $\mathrm{J}(\mathcal{A}, G_\lambda)$. By Lemma 3.6.2, this is equivalent to the pathfinder having a winning strategy in the dual game $\mathrm{J}(\bar{\mathcal{A}}, G_\lambda)$. By Lemma 3.5.7, this is the case if and only if the automaton does not have a winning strategy in $\mathrm{J}(\bar{\mathcal{A}}, G_\lambda)$. Again by Lemma 3.5.6, this is equivalent to saying that $\bar{\mathcal{A}}$ does not accept G_λ. Hence, $\mathcal{A}$ accepts G_λ if and only if $\bar{\mathcal{A}}$ does not accept G_λ. ■

Next, we prove closure under union and intersection. The following constructions exploit the power of nondeterminism and universal branching, and are, in principle, very similar to the corresponding constructions for alternating automata on words. However, because of the distributed nature of ADGAs, they are slightly more technical (local choices must be coordinated).

3.6.4 Lemma (Union and Intersection).
For every two ADGAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\langle \Sigma, \Gamma \rangle$, we can effectively construct ADGAs $\mathcal{A}_\cup$ and $\mathcal{A}_\cap$ that recognize the union language and intersection language, respectively, of $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e.,

$$\mathrm{L}(\mathcal{A}_\cup) = \mathrm{L}(\mathcal{A}_1) \cup \mathrm{L}(\mathcal{A}_2), \quad \text{and} \quad \mathrm{L}(\mathcal{A}_\cap) = \mathrm{L}(\mathcal{A}_1) \cap \mathrm{L}(\mathcal{A}_2).$$

Moreover,

$$\begin{aligned} \mathrm{siz}(\mathcal{A}_\cup) &= \mathrm{siz}(\mathcal{A}_\cap) = \mathrm{siz}(\mathcal{A}_1) + \mathrm{siz}(\mathcal{A}_2) + |\Sigma| + 1, \quad \text{and} \\ \mathrm{len}(\mathcal{A}_\cup) &= \mathrm{len}(\mathcal{A}_\cap) = \max\bigl\{\mathrm{len}(\mathcal{A}_1),\, \mathrm{len}(\mathcal{A}_2)\bigr\} + 1. \end{aligned}$$

Proof. Let $\mathcal{A}_1 = \langle \Sigma, \Gamma, \widehat{Q}_1, \sigma_1, \delta_1, \mathcal{F}_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, \Gamma, \widehat{Q}_2, \sigma_2, \delta_2, \mathcal{F}_2 \rangle$. Without loss of generality, we may assume that

- both automata are nonblocking and trim (see Remarks 3.4.2 and 3.4.4), and
- they agree on the sequence of quantifiers, i.e., for $0 \leq i < \min\{\mathrm{len}(\mathcal{A}_1), \mathrm{len}(\mathcal{A}_2)\}$, the state sets $\mathrm{lev}_i(\mathcal{A}_1)$ and $\mathrm{lev}_i(\mathcal{A}_2)$ are either both existential

or both universal, in the respective automata. By Remark 3.4.6, a simple way to ensure this is to transform both automata into alternating normal form, and possibly inserting an additional "dummy" level into one of them.

Further, let Q_Σ be a set of states with the same cardinality as Σ, where $q_a \in Q_\Sigma$ denotes the state corresponding to $a \in \Sigma$, and let q_{acc} and q_{rej} be two additional states. We assume that Q_1, Q_2, Q_Σ and $\{q_{\mathrm{acc}}, q_{\mathrm{rej}}\}$ are pairwise disjoint.

First, we construct the union automaton. The idea is that, in the first round, each node in the input graph nondeterministically and independently decides whether to behave like in $\mathcal{A}_1$ or in $\mathcal{A}_2$. If there is a consensus, then the run continues as it would in the unanimously chosen automaton $\mathcal{A}_j$, and it is accepting if and only if it corresponds to an accepting run of $\mathcal{A}_j$. Otherwise, a conflict is detected, either locally by adjacent nodes, or at the latest, when acceptance is checked globally, and in either case the run is rejecting. Formally, we define $\mathcal{A}_\cup = \langle \Sigma, \Gamma, \widehat{Q}_\cup, \sigma_\cup, \delta_\cup, \mathcal{F}_\cup \rangle$, where

- $(Q_\cup)_\exists = (Q_1)_\exists \cup (Q_2)_\exists \cup Q_\Sigma$,
- $(Q_\cup)_\forall = (Q_1)_\forall \cup (Q_2)_\forall$,
- $(Q_\cup)_{\mathrm{P}} = (Q_1)_{\mathrm{P}} \cup (Q_2)_{\mathrm{P}} \cup \{q_{\mathrm{rej}}\}$,
- $\sigma_\cup(a) = q_a, \quad$ for every $a \in \Sigma$,
- $$\delta_\cup(q, \widehat{S}) = \begin{cases} \{\sigma_1(a),\, \sigma_2(a)\} & \text{if } q = q_a \in Q_\Sigma \text{ and } \widehat{S} \in (2^{Q_\Sigma})^\Gamma, \\ \delta_1(q, \widehat{S}) & \text{if } q \in Q_1 \text{ and } \widehat{S} \in (2^{Q_1})^\Gamma, \\ \delta_2(q, \widehat{S}) & \text{if } q \in Q_2 \text{ and } \widehat{S} \in (2^{Q_2})^\Gamma, \\ \{q_{\mathrm{rej}}\} & \text{otherwise,} \end{cases}$$
  for every $q \in (Q_\cup)_{\mathrm{N}}$ and $\widehat{S} \in (2^{Q_\cup})^\Gamma$,
- $\mathcal{F}_\cup = \mathcal{F}_1 \cup \mathcal{F}_2$.

Note that $\delta_\cup$ satisfies the properties required by the definition of an ADGA, in particular, that states on the same level are in the same component of $\widehat{Q}_\cup$, which is guaranteed by the assumptions that $\mathcal{A}_1$ and $\mathcal{A}_2$ are both trim and agree on the sequence of quantifiers.

Next, we verify that, for any $G_\lambda \in \Sigma^{\textcircled{\varGamma}}$, $\mathcal{A}_\cup$ accepts G_λ if and only if $\mathcal{A}_1$ or $\mathcal{A}_2$ accepts G_λ.

($\Leftarrow$) Let one of the two automata, say $\mathcal{A}_j$, $j \in \{1, 2\}$, have an accepting run $R = \langle K, \rightharpoonup \rangle$ on G_λ. By construction, the initial configuration of $\mathcal{A}_\cup$ on G_λ is existential, and there is a global transition to the initial configuration of $\mathcal{A}_j$ on G_λ, i.e., $G_{\sigma_j \circ \lambda} \in \delta_\cup^{\circledast}(G_{\sigma_\cup \circ \lambda})$. Moreover, any transition of $\mathcal{A}_j$ is also a transition of $\mathcal{A}_\cup$, and configurations common to $\mathcal{A}_j$ and $\mathcal{A}_\cup$ have the same type (e.g., existential, etc.) in both automata. Thus,

$$R' = \big\langle K \cup \{G_{\sigma_\cup \circ \lambda}\},\ \rightharpoonup \cup \{\langle G_{\sigma_\cup \circ \lambda}, G_{\sigma_j \circ \lambda} \rangle\} \big\rangle$$

is a run of $\mathcal{A}_\cup$ on G_λ, and since $\mathcal{F}_j \subseteq \mathcal{F}_\cup$, any permanent configuration occurring in R' is accepting, which entails that R' is also accepting.

($\Rightarrow$) Now let $\mathcal{A}_\cup$ have an accepting run R' on G_λ. Since the initial configuration of $\mathcal{A}_\cup$ on G_λ is existential, it must have exactly one outgoing neighbor

G_κ in R'. By construction of $\mathcal{A}_\cup$, every state occurring in G_κ belongs to either $\mathcal{A}_1$ or $\mathcal{A}_2$, i.e., $G_\kappa \in (Q_1 \cup Q_2)^G$.

Assume that states of both automata occur in G_κ. We generalize this property by calling any configuration G_μ of $\mathcal{A}_\cup$ *impure* if $G_\mu \in (Q_1 \cup Q_2 \cup \{q_{\mathrm{rej}}\})^G \setminus (Q_1^G \cup Q_2^G)$.

- Any successor configuration of an impure configuration is also impure, because there are no local transitions from $Q_2 \cup \{q_{\mathrm{rej}}\}$ to Q_1, or from $Q_1 \cup \{q_{\mathrm{rej}}\}$ to Q_2.
- Further, any impure permanent configuration G_μ is rejecting, because G_μ being impure means that $\{\mu(v) \mid v \in V_G\} \notin (2^{Q_1} \cup 2^{Q_2})$, and consequently the acceptance condition given by $\mathcal{F}_1 \cup \mathcal{F}_2$ cannot be fulfilled.

Hence, the above assumption implies that all the permanent configurations among G_κ and its descendant configurations under $(\delta_\cup)^{✩}$ are rejecting. Since we required $\mathcal{A}_1$ and $\mathcal{A}_2$ to be nonblocking, such permanent configurations must exist (every nonpermanent configuration has at least one successor configuration). It follows that R' is not accepting, which is a contradiction.

We conclude that only states of one automaton, say $\mathcal{A}_j,\ j \in \{1,2\}$, can occur in G_κ. More precisely, G_κ is the initial configuration of $\mathcal{A}_j$ on G_λ, i.e., $G_\kappa = G_{\sigma_j \circ \lambda}$. Since $\mathcal{A}_\cup$ behaves like $\mathcal{A}_j$ on configurations of $\mathcal{A}_j$, this means that R' is exactly of the same form as the run constructed in the previous part of this proof, i.e.,

$$R' = \big\langle \{G_{\sigma_\cup \circ \lambda}\} \mathbin{\dot\cup} K,\ \ \{\langle G_{\sigma_\cup \circ \lambda}, G_{\sigma_j \circ \lambda}\rangle\} \mathbin{\dot\cup} \rightharpoonup \big\rangle,$$

where $K \subseteq Q_j^G$ and $\rightharpoonup\, \subseteq K \times K$. By removing the initial configuration (and its outgoing edge), we get a run $R = \langle K, \rightharpoonup\rangle$ of $\mathcal{A}_j$ on G_λ, which is accepting because $\mathcal{F}_\cup \cap 2^{Q_j} = \mathcal{F}_j$.

Finally, we turn our attention to the intersection automaton. By De Morgan's law, $\overline{\mathrm{L}(\mathcal{A}_1) \cap \mathrm{L}(\mathcal{A}_2)} = \overline{\mathrm{L}(\mathcal{A}_1)} \cup \overline{\mathrm{L}(\mathcal{A}_2)}$, hence we can simply combine the available constructions for complementation and union, which leads to the intersection automaton $\mathcal{A}_\cap = \langle \Sigma, \Gamma, \widehat{Q}_\cap, \sigma_\cap, \delta_\cap, \mathcal{F}_\cap\rangle$, where

- $(Q_\cap)_{\exists} = (Q_1)_{\exists} \cup (Q_2)_{\exists}$,
- $(Q_\cap)_{\forall} = (Q_1)_{\forall} \cup (Q_2)_{\forall} \cup Q_\Sigma$,
- $(Q_\cap)_{\mathrm{P}} = (Q_1)_{\mathrm{P}} \cup (Q_2)_{\mathrm{P}} \cup \{q_{\mathrm{acc}}\}$,
- $\sigma_\cap(a) = q_a, \quad$ for every $a \in \Sigma$,
- $\delta_\cap(q, \widehat{S}) = \begin{cases} \{\sigma_1(a),\, \sigma_2(a)\} & \text{if } q = q_a \in Q_\Sigma \text{ and } \widehat{S} \in (2^{Q_\Sigma})^\Gamma, \\ \delta_1(q, \widehat{S}) & \text{if } q \in Q_1 \text{ and } \widehat{S} \in (2^{Q_1})^\Gamma, \\ \delta_2(q, \widehat{S}) & \text{if } q \in Q_2 \text{ and } \widehat{S} \in (2^{Q_2})^\Gamma, \\ \{q_{\mathrm{acc}}\} & \text{otherwise,} \end{cases}$

  for every $q \in (Q_\cap)_{\mathrm{N}}$ and $\widehat{S} \in (2^{Q_\cap})^\Gamma$,
- $\mathcal{F}_\cap = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \big(2^{(Q_\cap)_{\mathrm{P}}} \setminus (2^{(Q_1)_{\mathrm{P}}} \cup 2^{(Q_2)_{\mathrm{P}}})\big)$. ■

As a last type of operation on graph languages, we consider uniform relabelings of nodes, which we call node projections and formally define as follows.

3.6.5 Definition (Projection).
Let Σ and Σ' be two nonempty node alphabets and Γ an edge alphabet. A (node) projection from Σ to Σ' is a mapping $h\colon \Sigma \to \Sigma'$. With a slight abuse of notation, this mapping is extended to labeled graphs by applying it to each node label, and to graph languages by applying it to each labeled graph. More precisely, for every $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ and $L \subseteq \Sigma^{\textcircled{\Gamma}}$,

$$h(G_\lambda) := G_{h\circ\lambda}, \quad \text{and} \quad h(L) := \{h(G_\lambda) \mid G_\lambda \in L\}.$$

Again exploiting the power of nondeterminism, we can easily show that ADGA-recognizable graph languages are closed under arbitrary projections.

3.6.6 Lemma (Projection).
For every ADGA $\mathcal{A}$ over $\langle \Sigma, \Gamma \rangle$ and projection $h\colon \Sigma \to \Sigma'$, we can effectively construct an ADGA $\mathcal{A}'$ that recognizes the projected language of $\mathcal{A}$ through h, i.e.,

$$\mathrm{L}(\mathcal{A}') = h(\mathrm{L}(\mathcal{A})).$$

Moreover,

$$\mathrm{siz}(\mathcal{A}') = \mathrm{siz}(\mathcal{A}) + |\Sigma'| \quad \text{and} \quad \mathrm{len}(\mathcal{A}') = \mathrm{len}(\mathcal{A}) + 1.$$

Proof. The idea is simple: 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 $\mathcal{A}$ is simulated on that new input.

Without loss of generality, we may assume that $\mathcal{A}$ is trim (see Remark 3.4.4). Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$, and let $Q'_{\Sigma'}$ be a set of states with the same cardinality as Σ' and disjoint from Q, where $q_b \in Q'_{\Sigma'}$ denotes the state corresponding to $b \in \Sigma'$. We construct the projection automaton $\mathcal{A}' = \big\langle \Sigma', \Gamma, \langle Q'_{\exists}, Q_{\forall}, Q_{\mathrm{P}} \rangle, \sigma', \delta', \mathcal{F} \big\rangle$, where

- $Q'_{\exists} = Q_{\exists} \cup Q'_{\Sigma'}$,
- $\sigma'(b) = q_b, \quad$ for every $b \in \Sigma'$,
- $$\delta'(q, \widehat{S}) = \begin{cases} \{\sigma(a) \mid h(a) = b\} & \text{if } q = q_b \in Q'_{\Sigma'} \text{ and } \widehat{S} \in (2^{Q'_{\Sigma'}})^{\Gamma}, \\ \delta(q, \widehat{S}) & \text{if } q \in Q \text{ and } \widehat{S} \in (2^{Q})^{\Gamma}, \\ \emptyset & \text{otherwise}, \end{cases}$$

  for every $q \in (Q_{\mathrm{N}} \cup Q'_{\Sigma'})$ and $\widehat{S} \in (2^{Q \cup Q'_{\Sigma'}})^{\Gamma}$.

Note that, as required, states on the same level of $\mathcal{A}'$ are of the same type (e.g., existential, etc.), because we have assumed that $\mathcal{A}$ is trim. Consider any $G_{\lambda'} \in \Sigma'^{\textcircled{\Gamma}}$. The initial configuration $G_{\sigma'\circ\lambda'}$ of $\mathcal{A}'$ on $G_{\lambda'}$ is existential. Its successor configurations are the initial configurations of $\mathcal{A}$ on the Σ-labeled Γ-graphs that are mapped to $G_{\lambda'}$ by h, i.e.,

$$\delta'^{\pentagon}(G_{\sigma'\circ\lambda'}) = \big\{ G_{\sigma\circ\lambda} \,\big|\, G_\lambda \in \Sigma^{G} \wedge h(G_\lambda) = G_{\lambda'} \big\}.$$

Moreover, the behaviours of $\mathcal{A}$ and $\mathcal{A}'$ on configurations of $\mathcal{A}$ are the same. Hence, arguing similarly as for the union construction (Lemma 3.6.4), we can show that $\mathcal{A}'$ accepts $G_{\lambda'}$ if and only if $\mathcal{A}$ accepts some $G_\lambda \in \Sigma^G$ such that $h(G_\lambda) = G_{\lambda'}$. ■

The following theorem summarizes the closure properties stated in Lemmas 3.6.3, 3.6.4 and 3.6.6.

3.6.7 Theorem (Closure Properties).
The class $\mathcal{L}_{\mathrm{ADGA}}$ of ADGA-recognizable graph languages is effectively closed under boolean set operations and under projection.

Chapter 4

Monadic Second-Order Logic on Graphs

In this chapter, we review monadic second-order (MSO) logic on labeled graphs. Then, building on the results of Chapter 3, we prove our main result: the ADGA-recognizable graph languages are precisely the MSO-definable ones. This, in turn, allows us to infer some negative properties of ADGAs.

4.1 Basic Definitions

Throughout this work, we fix two disjoint, countably infinite sets of (object language) variables: the supply of node variables $\mathcal{V}_{\text{node}} = \{\mathsf{u}, \mathsf{v}, \dots, \mathsf{u}_1, \dots\}$, and the supply of set variables $\mathcal{V}_{\text{set}} = \{\mathsf{U}, \mathsf{V}, \dots, \mathsf{U}_1, \dots\}$. Node variables will always be represented by lower-case letters, and set variables by upper-case ones (sometimes with subscripts).[1]

4.1.1 Definition (MSO-Logic: Syntax).
Let Σ be a node alphabet and Γ an edge alphabet. The set $\mathrm{MSO}(\Sigma, \Gamma)$ of monadic second-order formulas (on graphs) over $\langle \Sigma, \Gamma \rangle$ is built up from the atomic formulas

- $\langle a \rangle x$ ("x has label a"),
- $x \xrightarrow{\gamma} y$ ("x has a γ-edge to y"),
- $x = y$ ("x is equal to y"),
- $x \in X$ ("x is an element of X"),

for all $x, y \in \mathcal{V}_{\text{node}}$, $X \in \mathcal{V}_{\text{set}}$, $a \in \Sigma$, and $\gamma \in \Gamma$, using the usual propositional connectives and quantifiers, which can be applied to both node and set variables. More precisely, if φ and ψ are $\mathrm{MSO}(\Sigma, \Gamma)$-formulas, then so are $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \Rightarrow \psi$, $\varphi \Leftrightarrow \psi$, $\exists x(\varphi)$, $\forall x(\varphi)$, $\exists X(\varphi)$, and $\forall X(\varphi)$, for all $x \in \mathcal{V}_{\text{node}}$ and $X \in \mathcal{V}_{\text{set}}$.

[1] Concrete instances of such object language variables will be typeset in a bright blue sans-serif font, to better distinguish them from meta-language variables, which can refer (amongst others) to arbitrary object language variables.

We will consistently represent MSO-formulas in the typographic style used above, to distinguish object language from meta-language.[2]

An occurrence of a variable $x \in \mathcal{V}_{\text{node}}$ or $X \in \mathcal{V}_{\text{set}}$ in a formula φ is said to be free if it is not within the scope of a quantifier. We denote by $\text{free}(\varphi)$ the set of variables that occur freely in φ. If φ has no free occurrences of variables, i.e., if $\text{free}(\varphi) = \emptyset$, we say that φ is a sentence. Moreover, we will 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., $\text{free}(\varphi) \subseteq \{x_1, \dots, x_m, X_1, \dots, X_n\}$. This notation will also occasionally be abused to instantiate a formula (schema) with concrete variables.[3]

4.1.2 Definition (MSO-Logic: Semantics).
The truth of an $\text{MSO}(\Sigma, \Gamma)$-formula φ is evaluated with respect to a labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ and a variable assignment $\alpha\colon \text{free}(\varphi) \to V_G \cup 2^{V_G}$ that assigns a node $v \in V_G$ to each node variable in $\text{free}(\varphi)$, and a set of nodes $S \subseteq V_G$ to each set variable in $\text{free}(\varphi)$. We write $\langle G_\lambda, \alpha\rangle \models \varphi$ to denote that G_λ and α satisfy φ. If φ is a sentence, the variable assignment is superfluous, and we simply write $G_\lambda \models \varphi$ if G_λ satisfies φ. The meaning of the atomic formulas is as hinted informally in Definition 4.1.1, i.e.,

- $\langle G_\lambda, \alpha\rangle \models \langle a\rangle x \quad \Leftrightarrow \quad \lambda(\alpha(x)) = a$,
- $\langle G_\lambda, \alpha\rangle \models x \overset{\gamma}{\rightharpoonup} y \quad \Leftrightarrow \quad \alpha(x) \overset{\gamma}{\rightharpoonup}_G \alpha(y)$,
- $\langle G_\lambda, \alpha\rangle \models x = y \quad \Leftrightarrow \quad \alpha(x) = \alpha(y)$,
- $\langle G_\lambda, \alpha\rangle \models x \in X \quad \Leftrightarrow \quad \alpha(x) \in \alpha(X)$.

for all $x, y \in \mathcal{V}_{\text{node}}$, $X \in \mathcal{V}_{\text{set}}$, $a \in \Sigma$, and $\gamma \in \Gamma$. Composed formulas are interpreted according to the usual semantics of second-order logic, i.e.,

- $\langle G_\lambda, \alpha\rangle \models \neg\varphi \quad \Leftrightarrow \quad \langle G_\lambda, \alpha\rangle \not\models \varphi$,
- $\langle G_\lambda, \alpha\rangle \models \varphi \vee \psi \quad \Leftrightarrow \quad \langle G_\lambda, \alpha\rangle \models \varphi$ or $\langle G_\lambda, \alpha\rangle \models \psi$,
- $\langle G_\lambda, \alpha\rangle \models \varphi \wedge \psi \quad \Leftrightarrow \quad \langle G_\lambda, \alpha\rangle \models \varphi$ and $\langle G_\lambda, \alpha\rangle \models \psi$,
- $\langle G_\lambda, \alpha\rangle \models \varphi \Rightarrow \psi \quad \Leftrightarrow \quad \langle G_\lambda, \alpha\rangle \not\models \varphi$ or $\langle G_\lambda, \alpha\rangle \models \psi$,
- $\langle G_\lambda, \alpha\rangle \models \varphi \Leftrightarrow \psi \quad \Leftrightarrow \quad \langle G_\lambda, \alpha\rangle \models \varphi \Rightarrow \psi$ and $\langle G_\lambda, \alpha\rangle \models \psi \Rightarrow \varphi$,
- $\langle G_\lambda, \alpha\rangle \models \exists x(\varphi) \quad \Leftrightarrow \quad \langle G_\lambda, \alpha[x \mapsto v]\rangle \models \varphi$ for some $v \in V_G$,

[2] In order to make a clear distinction between MSO-formulas and formal statements at the meta-level (where some of the same symbols are used), the former will always be represented on a light blue background, using a bright blue font for symbols that directly occur in the considered formula. In contrast to this, other symbols must be interpreted at the meta-level to get the intended MSO-formulas. As usual, notations like $\bigwedge_{1 \le i \le n} \varphi_i$ and $\exists_{1 \le i \le n} x_i(\varphi)$ are used to represent $\varphi_1 \wedge \dots \wedge \varphi_n$ and $\exists x_1(\cdots \exists x_n(\varphi))$, respectively.

[3] Strictly speaking, if $x \in \mathcal{V}_{\text{node}}$ and $X \in \mathcal{V}_{\text{set}}$ are unspecified, an object like $\varphi[x, X] = x \in X$ is a formula schema. We can instantiate it with concrete object language variables, for instance v and U, to obtain the formula $\mathsf{v} \in \mathsf{U}$, which (by slight abuse of notation) will be denoted by $\varphi[\mathsf{v}, \mathsf{U}]$. To simplify matters, we shall henceforth not explicitly distinguish formula schemata from formulas.

- $\langle G_\lambda, \alpha \rangle \models \forall x(\varphi) \quad \Leftrightarrow \quad \langle G_\lambda, \alpha[x \mapsto v] \rangle \models \varphi$ for all $v \in V_G$,
- $\langle G_\lambda, \alpha \rangle \models \exists X(\varphi) \quad \Leftrightarrow \quad \langle G_\lambda, \alpha[X \mapsto U] \rangle \models \varphi$ for some $U \subseteq V_G$,
- $\langle G_\lambda, \alpha \rangle \models \forall X(\varphi) \quad \Leftrightarrow \quad \langle G_\lambda, \alpha[X \mapsto U] \rangle \models \varphi$ for all $U \subseteq V_G$,

for all $\varphi, \psi \in \mathrm{MSO}(\Sigma, \Gamma)$, $x \in \mathcal{V}_{\mathrm{node}}$ and $X \in \mathcal{V}_{\mathrm{set}}$. Here, $\alpha[x \mapsto v]$ designates the variable assignment that coincides with α except for x, which is mapped to v, and analogously, $\alpha[X \mapsto U]$ coincides with α except for X, which is mapped to U.

We will omit some unnecessary parentheses by following some of the usual precedence rules for propositional connectives: $\neg$ binds stronger than $\vee$ and $\wedge$, which in turn bind stronger than $\Rightarrow$ and $\Leftrightarrow$.

4.1.3 Definition (MSO-Definability).
The graph language $\mathrm{L}_{\Sigma,\Gamma}(\varphi)$ defined by an $\mathrm{MSO}(\Sigma, \Gamma)$-sentence φ, with respect to $\langle \Sigma, \Gamma \rangle$, is the set of all Σ-labeled Γ-graphs for which the sentence is satisfied, i.e.,

$$\mathrm{L}_{\Sigma,\Gamma}(\varphi) := \{ G_\lambda \in \Sigma^{\textcircled{\scriptsize Γ}} \mid G_\lambda \models \varphi \}.$$

Every graph language that is defined by some MSO-sentence is called MSO-definable. We denote by $\mathcal{L}_{\mathrm{MSO}}$ the class of all such graph languages.

An $\mathrm{MSO}(\Sigma, \Gamma)$-sentence φ is equivalent to an ADGA $\mathcal{A}$ over $\langle \Sigma, \Gamma \rangle$ if it defines the same graph language as $\mathcal{A}$ recognizes, i.e., $\mathrm{L}_{\Sigma,\Gamma}(\varphi) = \mathrm{L}(\mathcal{A})$.

We now revisit two of the graph languages considered in Chapter 3, and define them by MSO-sentences.

4.1.4 Example (Translation of $\mathcal{A}_{\mathrm{centric}}$ to MSO-Logic).
We fix $\Sigma = \{a, b, c\}$ and $\Gamma = \{\square\}$. The following $\mathrm{MSO}(\Sigma, \Gamma)$-sentence $\varphi_{\mathrm{centric}}$ is equivalent to the ADGA $\mathcal{A}_{\mathrm{centric}}$ from Fig. 3.3 (see Example 3.2.8 for a discussion of the recognized graph language).

$$\begin{aligned}
\varphi_{\mathrm{centric}} := {} & \forall \mathsf{u}, \mathsf{v} \Big(\mathsf{u} \rightharpoonup \mathsf{v} \;\Rightarrow\; \neg(\langle b \rangle \mathsf{u} \wedge \langle b \rangle \mathsf{v}) \wedge \neg(\langle c \rangle \mathsf{u} \wedge \langle c \rangle \mathsf{v}) \Big) \;\wedge \\
& \exists \mathsf{v}_a \Big(\forall \mathsf{u} \Big((\langle a \rangle \mathsf{u} \Leftrightarrow \mathsf{u} = \mathsf{v}_a) \;\wedge\; (\mathsf{u} \rightharpoonup \mathsf{v}_a \vee \mathsf{v}_a \rightharpoonup \mathsf{u} \;\Rightarrow\; \langle b \rangle \mathsf{u}) \Big) \;\wedge \\
& \qquad\qquad \exists \mathsf{u}_1, \mathsf{u}_2 \Big(\mathsf{u}_1 \rightharpoonup \mathsf{v}_a \;\wedge\; \mathsf{u}_2 \rightharpoonup \mathsf{v}_a \;\wedge\; \neg\, \mathsf{u}_1 = \mathsf{u}_2 \Big) \Big)
\end{aligned}$$

The first line ensures that no two adjacent nodes are both b-labeled or both c-labeled. The other two lines specify the existence of the "center", a node v_a such that

- v_a is the only a-labeled node in the graph and has only b-labeled nodes in its undirected neighborhood (second line), and
- v_a has at least two distinct incoming neighbors (third line).

In the preceding example, we did not exploit the possibility of quantifying over set variables. This makes $\varphi_{\mathrm{centric}}$ a *first*-order formula. The next example (slightly adapted from [CE12]) shows an application that requires second-order quantification.

4.1.5 Example (3-Colorability).
Let $\Sigma = \Gamma = \{\square\}$. The following $\mathrm{MSO}(\Sigma,\Gamma)$-sentence $\varphi_3^{\mathrm{color}}$ defines (with respect to $\langle\Sigma,\Gamma\rangle$) the language of 3-colorable graphs. It is thus equivalent to the ADGA $\mathcal{A}_3^{\mathrm{color}}$ from Fig. 3.1.

$$\begin{aligned}\varphi_3^{\mathrm{color}} := \exists \mathsf{U}_\spadesuit, \mathsf{U}_\heartsuit, \mathsf{U}_\clubsuit \Big(& \forall \mathsf{u}\Big((\mathsf{u} \in \mathsf{U}_\spadesuit \vee \mathsf{u} \in \mathsf{U}_\heartsuit \vee \mathsf{u} \in \mathsf{U}_\clubsuit) \wedge \neg(\mathsf{u} \in \mathsf{U}_\spadesuit \wedge \mathsf{u} \in \mathsf{U}_\heartsuit) \wedge \\ & \neg(\mathsf{u} \in \mathsf{U}_\spadesuit \wedge \mathsf{u} \in \mathsf{U}_\clubsuit) \wedge \neg(\mathsf{u} \in \mathsf{U}_\heartsuit \wedge \mathsf{u} \in \mathsf{U}_\clubsuit)\Big) \wedge \\ & \forall \mathsf{u}, \mathsf{v}\Big(\mathsf{u} \rightharpoonup \mathsf{v} \;\Rightarrow\; \neg(\mathsf{u} \in \mathsf{U}_\spadesuit \wedge \mathsf{v} \in \mathsf{U}_\spadesuit) \wedge \\ & \neg(\mathsf{u} \in \mathsf{U}_\heartsuit \wedge \mathsf{v} \in \mathsf{U}_\heartsuit) \wedge \neg(\mathsf{u} \in \mathsf{U}_\clubsuit \wedge \mathsf{v} \in \mathsf{U}_\clubsuit)\Big)\Big)\end{aligned}$$

The existentially quantified set variables $\mathsf{U}_\spadesuit$, $\mathsf{U}_\heartsuit$ and $\mathsf{U}_\clubsuit$ represent the three possible colors. In the first two lines, we specify that the sets assigned to these variables form a partition of the set of nodes (possibly with empty components). The remaining two 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.

Similarly to Examples 4.1.4 and 4.1.5, we could translate all the ADGAs seen in the examples of Chapter 3 to equivalent MSO-sentences. In the next section, we generalize this for arbitrary ADGAs.

4.2 Equivalence of MSO-Logic and ADGAs

We can now formally state and prove our main theorem.

4.2.1 Theorem ($\mathcal{L}_{\mathrm{ADGA}} = \mathcal{L}_{\mathrm{MSO}}$).
A graph language is ADGA-recognizable if and only if it is MSO-definable. There are effective translations in both directions.

We divide the proof of Theorem 4.2.1 into Lemmas 4.2.2 and 4.2.3, which correspond to the two directions of translation.

4.2.2 Lemma ($\mathcal{L}_{\mathrm{ADGA}} \subseteq \mathcal{L}_{\mathrm{MSO}}$).
For every ADGA $\mathcal{A}$ over $\langle\Sigma,\Gamma\rangle$, we can effectively construct an $\mathrm{MSO}(\Sigma,\Gamma)$-sentence $\varphi_{\mathcal{A}}$ that is equivalent to $\mathcal{A}$, i.e.,

$$\mathrm{L}_{\Sigma,\Gamma}(\varphi_{\mathcal{A}}) = \mathrm{L}(\mathcal{A}).$$

Proof. Let $\mathcal{A} = \langle\Sigma,\Gamma,\widehat{Q},\sigma,\delta,\mathcal{F}\rangle$. We have to construct an $\mathrm{MSO}(\Sigma,\Gamma)$-sentence $\varphi_{\mathcal{A}}$, such that any $G_\lambda \in \Sigma^{\textcircled{\scriptsize\Gamma}}$ satisfies $\varphi_{\mathcal{A}}$ if and only if it is accepted by $\mathcal{A}$. Thus, $\varphi_{\mathcal{A}}$ must somehow encode the existence of an accepting run of $\mathcal{A}$ on G_λ. A direct approach might seem tricky at first, since a run is a nontrivial graph itself, whose nodes are not in the domain of discourse that is referred to from within $\varphi_{\mathcal{A}}$. We can simplify the problem by taking again the game-theoretic point of view introduced in Section 3.5. By Lemma 3.5.6, $\mathcal{A}$ accepts G_λ if and only if the automaton has a winning strategy in the associated game $J = \mathrm{J}(\mathcal{A}, G_\lambda)$. We will exploit this equivalence, and construct an MSO-sentence expressing that the automaton can win the game, no matter how the pathfinder chooses to play.

Throughout this proof, we use the abbreviations $n := \mathrm{len}(\mathcal{A})$, for the length of the automaton, and $Q_i := \mathrm{lev}_i(\mathcal{A}) \cup Q_{\mathrm{P}}$, for the set of states that may occur in a configuration reachable by $\mathcal{A}$ in round i, where $0 \le i \le n$. Our sentence $\varphi_{\mathcal{A}}$ will contain the set variables $\mathsf{U}_{i,q} \in \mathcal{V}_{\mathrm{set}}$, for every round $i \ge 1$ and state $q \in Q_i$. The intended meaning of the subformula $\mathsf{v} \in \mathsf{U}_{i,q}$ is the following: given a prefix $\pi = G_{\kappa_0} \cdots G_{\kappa_i}$ of a play in J, the node $v \in V_G$ assigned to the variable $\mathsf{v} \in \mathcal{V}_{\mathrm{node}}$ is in state q in round i, i.e., $\kappa_i(v) = q$. An entire play will thus be represented by an assignment to the set variables $\mathsf{U}_{i,q}$. Note that we do not need set variables for round 0, since the initial configuration $G_{\kappa_0} = G_{\sigma\circ\lambda}$ is always the same. We abbreviate by $\widehat{X}_i$ the list of set variables for round i, i.e., $\widehat{X}_0 := \langle\,\rangle$ and $\widehat{X}_i := \langle \mathsf{U}_{i,q} \rangle_{q \in Q_i}$, for $1 \le i \le n$.

We now construct $\varphi_{\mathcal{A}}$ bottom-up, starting with the simple building block mentioned above. For $0 \le i \le n$, $q \in Q_i$, and $x \in \mathcal{V}_{\mathrm{node}}$, the subformulas $\varphi^{\mathrm{state}}_{i:q}[x, \widehat{X}_i]$ express that in round i, the node assigned to x is in state q. They are defined by

$$\varphi^{\mathrm{state}}_{i:q}[x, \widehat{X}_i] \;:=\; \begin{cases} \displaystyle\bigvee_{a \in \Sigma :\, \sigma(a) = q} \langle a \rangle x & \text{if } i = 0, \\[2ex] x \in \mathsf{U}_{i,q} & \text{if } 1 \le i \le n. \end{cases}$$

Next, for $0 \le i \le n$, $q \in Q_i$, and $\widehat{S} = \langle S_\gamma \rangle_{\gamma \in \Gamma} \in (2^{Q_i})^{\Gamma}$, the subformulas $\varphi^{\mathrm{neigh}}_{i:q,\widehat{S}}[\mathsf{v}, \widehat{X}_i]$ express that in round i, the node assigned to v receives the information $\langle q, \widehat{S} \rangle$ from its closed incoming neighborhood. This is ensured by checking that the local state is q, and for each $\gamma \in \Gamma$, every state in S_γ is seen on some γ-edge, and any state seen on a γ-edge is in S_γ:

$$\begin{aligned} \varphi^{\mathrm{neigh}}_{i:q,\widehat{S}}[\mathsf{v}, \widehat{X}_i] \;:=\; & \varphi^{\mathrm{state}}_{i:q}[\mathsf{v}, \widehat{X}_i] \;\wedge \bigwedge_{\gamma \in \Gamma,\, p \in S_\gamma} \exists \mathsf{u} \Big(\varphi^{\mathrm{state}}_{i:p}[\mathsf{u}, \widehat{X}_i] \wedge \mathsf{u} \xrightarrow{\gamma} \mathsf{v} \Big) \\ & \wedge \forall \mathsf{u} \Big(\bigwedge_{\gamma \in \Gamma} \Big(\mathsf{u} \xrightarrow{\gamma} \mathsf{v} \Rightarrow \bigvee_{p \in S_\gamma} \varphi^{\mathrm{state}}_{i:p}[\mathsf{u}, \widehat{X}_i] \Big) \Big). \end{aligned}$$

With building blocks for these local properties at our disposal, we can now proceed to more global statements. For the remainder of this proof, we set $G_{\kappa_0} = G_{\sigma\circ\lambda}$, and for $1 \le i \le n$, we refer by G_{κ_i} to the configuration represented by some given assignment to the set variables in $\widehat{X}_i$.

The meaning of the subformula $\varphi^{\mathrm{legal}}_i[\widehat{X}_{i-1}, \widehat{X}_i]$ is that G_{κ_i} is a legal successor configuration of $G_{\kappa_{i-1}}$. Two properties have to be checked: On the one hand, it must hold that $G_{\kappa_i} \in \delta^{\text{✿}}(G_{\kappa_{i-1}})$, or equivalently, that for every node $v \in V_G$, if v receives the information $\langle p, \widehat{S} \rangle$ in round $i-1$, then it is in some state $q \in \delta(p, \widehat{S})$ in round i. On the other hand, the given assignment to the set variables in $\widehat{X}_i$ must indeed represent a valid configuration, which in particular means that a node cannot be in several states at once. This leads to the definition

$$\varphi_i^{\mathrm{legal}}[\widehat{X}_{i-1}, \widehat{X}_i] \;\coloneqq\; \forall \mathsf{v}\Bigg(\bigwedge_{\substack{p \in Q_{i-1},\\ \widehat{S} \in (2^{Q_{i-1}})^{\Gamma}}} \Big(\varphi^{\mathrm{neigh}}_{i-1\,:\,p,\widehat{S}}[\mathsf{v}, \widehat{X}_{i-1}] \Rightarrow \bigvee_{q \in \delta(p,\widehat{S})} \varphi^{\mathrm{state}}_{i\,:\,q}[\mathsf{v}, \widehat{X}_i] \Big) \;\wedge\; \bigwedge_{q,r \in Q_i\,:\; q \neq r} \neg \Big(\varphi^{\mathrm{state}}_{i\,:\,q}[\mathsf{v}, \widehat{X}_i] \,\wedge\, \varphi^{\mathrm{state}}_{i\,:\,r}[\mathsf{v}, \widehat{X}_i] \Big) \Bigg).$$

We now come to our goal of expressing that the automaton has a winning strategy in J. As in the proof of Lemma 3.5.7, we consider every position G_κ in J as the starting position of a subgame J_κ, consisting of G_κ and all its descendant configurations. For every round i, we construct a subformula $\varphi_i^{\mathrm{win}}[\widehat{X}_i]$ expressing that the automaton has a winning strategy in the subgame J_{κ_i}.

In the last round n, the reached configuration can only be permanent, i.e., $G_{\kappa_n} \in (Q_{\mathrm{P}})^{\text{❀}\Gamma}$. Hence, the automaton has a winning strategy in J_{κ_n} if and only if G_{κ_n} is accepting. We check that there is an accepting set $F \in \mathcal{F}$, such that each state $q \in F$ occurs in G_{κ_n}, and only such states occur:

$$\varphi_n^{\mathrm{win}}[\widehat{X}_n] \;\coloneqq\; \bigvee_{F \in \mathcal{F}} \Bigg(\bigwedge_{q \in F} \exists \mathsf{v} \Big(\varphi^{\mathrm{state}}_{n\,:\,q}[\mathsf{v}, \widehat{X}_n] \Big) \;\wedge\; \forall \mathsf{v} \Big(\bigvee_{q \in F} \varphi^{\mathrm{state}}_{n\,:\,q}[\mathsf{v}, \widehat{X}_n] \Big) \Bigg).$$

Working our way backwards, we recursively define the formulas for previous rounds $i-1$, where $n \geq i \geq 1$. We have to distinguish two cases. If level $i-1$ is existential in $\mathcal{A}$, then the automaton is the player who has to make a move from position $G_{\kappa_{i-1}}$. Thus, it has a winning strategy in $J_{\kappa_{i-1}}$ if and only if there is a legal successor configuration G_{κ_i} of $G_{\kappa_{i-1}}$, for which the automaton has a winning strategy in the corresponding subgame J_{κ_i}. This is expressed by

$$\varphi_{i-1}^{\mathrm{win}}[\widehat{X}_{i-1}] \;\coloneqq\; \exists \widehat{X}_i \Big(\varphi_i^{\mathrm{legal}}[\widehat{X}_{i-1}, \widehat{X}_i] \;\wedge\; \varphi_i^{\mathrm{win}}[\widehat{X}_i] \Big).$$

Otherwise, level $i-1$ is universal, which means that the pathfinder has to make a move. Then the automaton has a winning strategy in $J_{\kappa_{i-1}}$ if and only if it has a winning strategy in every subgame that starts at a legal successor configuration of $G_{\kappa_{i-1}}$. The corresponding formula is analogous to the previous one:

$$\varphi_{i-1}^{\mathrm{win}}[\widehat{X}_{i-1}] \;\coloneqq\; \forall \widehat{X}_i \Big(\varphi_i^{\mathrm{legal}}[\widehat{X}_{i-1}, \widehat{X}_i] \;\Rightarrow\; \varphi_i^{\mathrm{win}}[\widehat{X}_i] \Big).$$

Note that these formulas also cover the cases where a permanent configuration is reached earlier than round n. If G_{κ_i} is permanent, for some $i < n$, then $\delta^{\text{❀}}(G_{\kappa_i}) = \{G_{\kappa_i}\}$, which means that $\varphi_{i+1}^{\mathrm{legal}}[\widehat{X}_i, \widehat{X}_{i+1}]$ is satisfied precisely when $\kappa_{i+1} = \kappa_i$. Proceeding inductively, we get that, regardless of whether level i is existential or universal, $\varphi_i^{\mathrm{win}}[\widehat{X}_i]$ is satisfied if and only if $\varphi_n^{\mathrm{win}}[\widehat{X}_n]$ is satisfied when interpreting the set variables in $\widehat{X}_n$ such that $\kappa_n = \kappa_i$. In other words, plays of length less than n are implicitly extended to length n by repeating the last configuration, and consequently the acceptance condition is always checked using the subformula $\varphi_n^{\mathrm{win}}[\widehat{X}_n]$.

We have thus achieved our goal. Since the subgame J_{κ_0} is equal to $\mathrm{J}(\mathcal{A}, G_\lambda)$, the desired MSO-sentence is

$$\varphi_{\mathcal{A}} := \varphi_0^{\text{win}}[\widehat{X}_0] = \varphi_0^{\text{win}}[\,]. \qquad \blacksquare$$

In [Eng91], Engelfriet characterized the class of MSO-definable graph languages as the smallest class that contains certain elementary graph languages[(4)] and is closed under boolean set operations and under projection. His elementary graph languages can be easily recognized by ADGAs, and thus, together with Theorem 3.6.7 and Lemma 4.2.2, this characterization of $\mathcal{L}_{\text{MSO}}$ implies our main theorem. Nevertheless, we give a self-contained proof of the following lemma, in order to provide a direct translation from MSO-formulas to ADGAs. Some of the ideas are adapted from [Eng91].

4.2.3 Lemma ($\mathcal{L}_{\text{ADGA}} \supseteq \mathcal{L}_{\text{MSO}}$).
For every $\text{MSO}(\Sigma, \Gamma)$-sentence φ, we can effectively construct an ADGA $\mathcal{A}_\varphi$ over $\langle \Sigma, \Gamma \rangle$ that is equivalent to φ, i.e.,

$$\mathrm{L}(\mathcal{A}_\varphi) = \mathrm{L}_{\Sigma,\Gamma}(\varphi).$$

Proof. It seems natural to prove the claim by induction on the structure of $\text{MSO}(\Sigma, \Gamma)$-formulas. This forces us to deal with formulas containing free occurrences of variables. The truth of such a formula φ is evaluated with respect to a labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ and a variable assignment $\alpha\colon \text{free}(\varphi) \to V_G \cup 2^{V_G}$. We have thus to represent $\langle G_\lambda, \alpha \rangle$ as a valid input for an ADGA. This can be done by encoding α into the node labels. To this end, we define the inverse function α^{-1} as the labeling that assigns to each node $v \in V_G$ the set of variables that α associates with v, i.e.,

$$\begin{aligned} \alpha^{-1}\colon\ & V_G \to 2^{\text{free}(\varphi)} \\ & v \mapsto \bigl\{x \in \mathcal{V}_{\text{node}} \bigm| v = \alpha(x)\bigr\} \cup \bigl\{X \in \mathcal{V}_{\text{set}} \bigm| v \in \alpha(X)\bigr\}. \end{aligned}$$

With this, $\langle G_\lambda, \alpha \rangle$ can be represented as the labeled graph $G_{\lambda \times \alpha^{-1}}$ whose labeling is given by

$$\begin{aligned} \lambda \times \alpha^{-1}\colon\ & V_G \to \Sigma \times 2^{\text{free}(\varphi)} \\ & v \mapsto \bigl\langle \lambda(v),\ \alpha^{-1}(v) \bigr\rangle. \end{aligned}$$

Using this encoding, we generalize the claim of the lemma as follows: For any $\text{MSO}(\Sigma, \Gamma)$-formula φ, there is an effectively constructible ADGA $\mathcal{A}_\varphi$, such that for every $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ and variable assignment $\alpha\colon \text{free}(\varphi) \to V_G \cup 2^{V_G}$,

$$G_{\lambda \times \alpha^{-1}} \in \mathrm{L}(\mathcal{A}_\varphi) \quad \text{if and only if} \quad \langle G_\lambda, \alpha \rangle \models \varphi.$$

If φ is a sentence, i.e., if $\text{free}(\varphi) = \emptyset$, we identify $G_{\lambda \times \alpha^{-1}}$ and $\langle G_\lambda, \alpha \rangle$ with G_λ. Hence, the statement above does indeed imply the lemma.

We now prove the generalized claim by structural induction on φ. In each case, we construct a suitable ADGA $\mathcal{A}_\varphi = \langle \Sigma \times 2^{\text{free}(\varphi)}, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$.

(4) According to Engelfriet's definition, a graph language L is elementary if and only if there are nonempty finite alphabets Σ and Γ, such that either $L = \Sigma^{\textcircled{\Gamma}}$, or $L = \bigl\{G_\lambda \in \Sigma^{\textcircled{\Gamma}} \bigm| \exists u, v \in V_G\colon \lambda(u) = a \wedge u \xrightarrow{\gamma}_G v \wedge \lambda(v) = b\bigr\}$, for some fixed $a, b \in \Sigma$ and $\gamma \in \Gamma$.

(BC) We start with the base cases, in which φ is an atomic formula.

For $b \in \Sigma$, $x, y \in \mathcal{V}_{\text{node}}$ and $X \in \mathcal{V}_{\text{set}}$, the truth of the formulas $\langle b \rangle x$, $x = y$ and $x \in X$ can be evaluated locally by an ADGA, i.e., without communication between the nodes. If φ is equal to such a formula, we define the states of $\mathcal{A}_\varphi$ as

$$Q_\exists = \emptyset, \quad Q_\forall = \emptyset \quad \text{and} \quad Q_\mathrm{P} = \{q_{\text{yes}}, q_{\text{no}}, q_{\text{maybe}}\}.$$

The intention here is that the node assigned to x (or y) will answer by "yes" or "no", while the other nodes remain undecided. The automaton then accepts the input if and only if the affected node answers positively, i.e.,

$$\mathcal{F} = \big\{\{q_{\text{yes}}\}, \{q_{\text{yes}}, q_{\text{maybe}}\}\big\}.$$

Since all the states are permanent, the transition function is already defined implicitly. It remains to specify, for each case, the initialization function which directly computes the answer of each node. For every $\langle a, M\rangle \in \Sigma \times 2^{\text{free}(\varphi)}$,

– if $\varphi = \langle b \rangle x$, then
$$\sigma(\langle a, M\rangle) = \begin{cases} q_{\text{yes}} & \text{if } a = b \text{ and } M = \{x\}, \\ q_{\text{no}} & \text{if } a \neq b \text{ and } M = \{x\}, \\ q_{\text{maybe}} & \text{otherwise}, \end{cases}$$

– if $\varphi = x = y$, then
$$\sigma(\langle a, M\rangle) = \begin{cases} q_{\text{yes}} & \text{if } M = \{x, y\}, \\ q_{\text{no}} & \text{if } M = \{x\} \text{ or } M = \{y\}, \text{ with } x \neq y, \\ q_{\text{maybe}} & \text{otherwise}, \end{cases}$$

– if $\varphi = x \in X$, then
$$\sigma(\langle a, M\rangle) = \begin{cases} q_{\text{yes}} & \text{if } M = \{x, X\}, \\ q_{\text{no}} & \text{if } M = \{x\}, \\ q_{\text{maybe}} & \text{otherwise}. \end{cases}$$

The last possible base case is when $\varphi = x \xrightarrow{\tau} y$, with $x, y \in \mathcal{V}_{\text{node}}$ and $\tau \in \Gamma$. To evaluate the truth of such a formula, an ADGA needs one communication round, after which the node assigned to y can check whether it has received a message from the node assigned to x through a τ-edge. Then, each node gives a local answer, and acceptance is decided as in the previous cases. Accordingly, we define the components of $\mathcal{A}_\varphi$ as follows:

– $Q_\exists = \{q_x, q_y, q_{x,y}\}, \quad Q_\forall = \emptyset, \quad Q_\mathrm{P} = \{q_{\text{yes}}, q_{\text{no}}, q_{\text{maybe}}\}$,

– $\sigma(\langle a, M\rangle) = \begin{cases} q_x & \text{if } M = \{x\}, \\ q_y & \text{if } M = \{y\}, \\ q_{x,y} & \text{if } M = \{x, y\}, \text{ with } x \neq y, \\ q_{\text{maybe}} & \text{otherwise}, \end{cases}$

  for every $\langle a, M\rangle \in \Sigma \times 2^{\{x,y\}}$,

– $\delta(q, \widehat{S}) = \begin{cases} \{q_{\text{maybe}}\} & \text{if } q = q_x, \\ \{q_{\text{yes}}\} & \text{if } \ q = q_y \ \text{ and } \ q_x \in S_\tau, \\ & \text{or } q = q_{x,y} \text{ and } q_{x,y} \in S_\tau, \\ \{q_{\text{no}}\} & \text{otherwise} \end{cases}$

for every $q \in Q_{\mathrm{N}}$ and $\widehat{S} = \langle S_\gamma \rangle_{\gamma \in \Gamma} \in (2^Q)^\Gamma$,

– $\mathcal{F} = \big\{\{q_{\text{yes}}\}, \{q_{\text{yes}}, q_{\text{maybe}}\}\big\}$.

Note that the transition function is deterministic, hence the choice of whether a nonpermanent state is existential or universal is arbitrary.

(IS) We now turn to the induction step, for which most of the work has already been done by proving the closure properties of ADGA-recognizable graph languages (Theorem 3.6.7). In the following, let ψ, ψ_1 and ψ_2 be $\mathrm{MSO}(\Sigma, \Gamma)$-formulas that satisfy the induction hypothesis with the ADGAs $\mathcal{A}_\psi$, $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, respectively.

If $\varphi = \neg\psi$, by Lemma 3.6.3, it suffices to define $\mathcal{A}_\varphi = \overline{\mathcal{A}}_\psi$.

Similarly, if $\varphi = \psi_1 \vee \psi_2$, we can use the union construction from Lemma 3.6.4. However, we must be careful because that construction can only be applied on automata that share the same node alphabet. If $\mathrm{free}(\psi_1) \neq \mathrm{free}(\psi_2)$, we have to extend the node alphabets and initialization functions of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, such that each automaton ignores the MSO-variables that are only relevant to the other one (as opposed to simply rejecting any input graph that contains unknown symbols). For instance, if $\langle a, M \rangle$ is a node label, and $x \in \mathrm{free}(\psi_2) \setminus \mathrm{free}(\psi_1)$, then the extended version of $\mathcal{A}_{\psi_1}$ will initialize a node labeled with $\langle a, M \cup \{x\} \rangle$ to the same state as one labeled with $\langle a, M \rangle$. The automaton $\mathcal{A}_\varphi$ is then obtained by applying the union construction on the extended versions of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$. We proceed analogously for the case where $\varphi = \psi_1 \wedge \psi_2$ (using the intersection construction from Lemma 3.6.4), and we reduce cases with other logical connectives to the previous ones.

Next, if $\varphi = \exists X(\psi)$, with $X \in \mathcal{V}_{\text{set}}$, we can take advantage of the projection construction from Lemma 3.6.6. An ADGA can evaluate the truth of φ by nondeterministically choosing which nodes are in the set assigned to X, and subsequently simulating $\mathcal{A}_\psi$. We thus construct $\mathcal{A}_\varphi$ by applying the projection construction on $\mathcal{A}_\psi$, using the mapping

$$\begin{aligned} h \colon \Sigma \times 2^{\mathrm{free}(\psi)} &\to \Sigma \times 2^{\mathrm{free}(\varphi) \setminus \{X\}} \\ \langle a, M \rangle &\mapsto \langle a, M \setminus \{X\} \rangle. \end{aligned}$$

Note that this also works if $X \notin \mathrm{free}(\psi)$, since then h is an identity function, and consequently $\mathrm{L}(\mathcal{A}_\varphi) = \mathrm{L}(\mathcal{A}_\psi)$, as required.

Some additional work is needed for the related case in which $\varphi = \exists x(\psi)$, with $x \in \mathcal{V}_{\text{node}}$. Like in the previous case, a corresponding ADGA can nondeterministically choose for each node whether or not it is assigned to x. But afterwards, it must check that precisely one node has been assigned to that variable. We construct a separate ADGA $\mathcal{A}_x^{\text{one}}$, specifically for the latter task, and then use it as a building block for $\mathcal{A}_\varphi$.

The idea is that any node assigned to x can universally choose between two colors. The automaton then accepts if and only if exactly one color has been chosen in each universal branch. Formally, we define $\mathcal{A}_x^{\mathrm{one}} = \langle \Sigma \times 2^{\mathrm{free}(\psi)}, \Gamma, \widehat{Q}_1, \sigma_1, \delta_1, \mathcal{F}_1 \rangle$, where

- $(Q_1)_\exists = \emptyset$, $\quad (Q_1)_\forall = \{q_x\}$, $\quad (Q_1)_\mathrm{P} = \{q_{\neg x}, q_\spadesuit, q_\heartsuit\}$,
- $\sigma_1(\langle a, M \rangle) = \begin{cases} q_x & \text{if } x \in M, \\ q_{\neg x} & \text{otherwise,} \end{cases}$ $\quad$ for every $\langle a, M \rangle \in \Sigma \times 2^{\mathrm{free}(\psi)}$,
- $\delta_1(q_x, \widehat{S}) = \{q_\spadesuit, q_\heartsuit\}$, $\quad$ for every $\widehat{S} \in (2^{Q_1})^\Gamma$,
- $\mathcal{F}_1 = \big\{ F \subseteq (Q_1)_\mathrm{P} \bigm| q_\spadesuit \in F \Leftrightarrow q_\heartsuit \notin F \big\}$.

We can now assemble $\mathcal{A}_\varphi$ by first applying the intersection construction on $\mathcal{A}_\psi$ and $\mathcal{A}_x^{\mathrm{one}}$, and then the projection construction on the resulting automaton, just as in the previous case, with x taking the role of X.

Finally, quantifier duality obviously covers the two cases with universal quantifiers, e.g., the formula $\forall x(\psi)$ can be replaced by $\neg\exists x(\neg\psi)$. It is worth mentioning, however, that this indirect approach does not even involve a blow-up of the resulting automata, since complementation leaves states and transitions unchanged. ■

This concludes the proof of Theorem 4.2.1.

4.3 Negative Implications for ADGAs

We can now take advantage of the equivalence between MSO-logic and ADGAs to infer some negative results on ADGAs.

The satisfiability problem of MSO-logic is the question whether, for a given MSO(Σ, Γ)-sentence φ, there is a labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$ that satisfies φ, i.e., whether $\mathrm{L}_{\Sigma,\Gamma}(\varphi) \neq \emptyset$. As remarked in [Cou97, CE12], the undecidability of this problem follows directly from Trakhtenbrot's Theorem [Tra50], which states that it is undecidable whether a first-order sentence over a relational vocabulary with at least one binary relation symbol is *finitely* satisfiable (see, e.g., [Lib04, Thm 9.2]).

4.3.1 Theorem (Satisfiability Problem).
The satisfiability problem of MSO-logic (on finite graphs) is undecidable.

Together with Theorem 4.2.1, we directly obtain the following corollary concerning the emptiness problem of ADGAs. This problem is the question whether the graph language $\mathrm{L}(\mathcal{A})$ of a given ADGA $\mathcal{A}$ is empty.

4.3.2 Corollary (Emptiness Problem).
The emptiness problem of ADGAs is undecidable.

Furthermore, Theorem 4.2.1 allows us to state some graph properties that cannot be recognized by ADGAs. This is a restatement of a result on MSO-logic proven by Courcelle and Engelfriet in [CE12, Prp 5.13].

4.3.3 Lemma (ADGA-Unrecognizable Languages).
Let $\Sigma = \Gamma = \{\Box\}$. The following graph languages are *not* ADGA-recognizable:

- $L_{\text{Ham}} = \left\{G \in \Sigma^{\textcircled{\scriptsize$\Gamma$}} \;\middle|\; G \text{ has a Hamiltonian cycle}\right\}$,
- $L_{\text{match}} = \left\{G \in \Sigma^{\textcircled{\scriptsize$\Gamma$}} \;\middle|\; G \text{ has a perfect matching}\right\}$, and
- $L_{\text{morph}} = \left\{G \in \Sigma^{\textcircled{\scriptsize$\Gamma$}} \;\middle|\; G \text{ has a nontrivial automorphism}\right\}$.

Chapter 5

Nondeterministic and Deterministic DGAs

In this chapter, we consider restrictions on the definition of ADGAs. It turns out that forbidding universal branchings results in a loss of expressive power, and additionally forbidding nondeterministic choices leads to an even weaker class of graph automata. On the other hand, the emptiness problem becomes decidable, and some closure properties still hold. Furthermore, as a byproduct of our investigation, we obtain necessary conditions for recognizability by those weaker classes of graph automata, loosely similar to pumping lemmas.

5.1 Nondeterministic Distributed Graph Automata

We start by removing the possibility of universal branching.

5.1.1 Definition (Nondeterministic Distributed Graph Automaton).
A nondeterministic distributed graph automaton (NDGA) is an ADGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ that has no universal states, i.e., $Q_{\forall} = \emptyset$. We denote by $\mathcal{L}_{\mathrm{NDGA}}$ the class of all NDGA-recognizable graph languages.

Since the runs of NDGAs do not branch, we can represent them simply as sequences of configurations of the form $R = G_{\kappa_0} \cdots G_{\kappa_n}$, with $n \leq \mathrm{len}(\mathcal{A})$, where G_{κ_0} is the initial configuration, each $G_{\kappa_{i+1}}$ is a successor configuration of G_{κ_i}, for $0 \leq i \leq n-1$, and G_{κ_n} is a permanent configuration.

We will compare such sequences from the local point of view of individual nodes.

5.1.2 Definition (Local View).
Consider a sequence $R = G_{\kappa_0} \cdots G_{\kappa_n}$ of configurations of some NDGA $\mathcal{A}$ on an underlying graph G. For each node $v \in V_G$, we define the local view $R|_v$ of v in R as the sequence of informations that v receives from its closed incoming neighborhood at each position of R, i.e.,

$$R|_v := \langle q_0, \widehat{S}_0 \rangle \cdots \langle q_n, \widehat{S}_n \rangle,$$

where $q_i = \kappa_i(v)$, and $\widehat{S}_i = \big\langle \{ \kappa_i(u) \mid u \xrightarrow{\gamma}_G v \} \big\rangle_{\gamma \in \Gamma}$, for $0 \leq i \leq n$.

On several occasions, we will construct a new run from a given one, by ensuring that every local view in the new run also occurs in the old run. The following remark formalizes this approach.

5.1.3 Remark.
Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be an NDGA, G_λ a labeled graph in $\Sigma^{\circledast}$, and $R = G_{\kappa_0} \cdots G_{\kappa_n}$ a run of $\mathcal{A}$ on G_λ. Consider another labeled graph $G'_{\lambda'} \in \Sigma^{\circledast}$. Suppose we can construct a sequence $R' = G'_{\kappa'_0} \cdots G'_{\kappa'_n}$ of configurations of $\mathcal{A}$ on G', such that, for every node $v' \in V_{G'}$, there is a node $v \in V_G$ with the same label and local view, i.e.,

- $\lambda(v) = \lambda'(v')$, and
- $R|_v = R'|_{v'}$.

Then R' is a legal run of $\mathcal{A}$ on $G'_{\lambda'}$. Furthermore, if R is accepting and the states occurring in G_{κ_n} and $G'_{\kappa'_n}$ are the same, i.e., $\{\kappa_n(v) \mid v \in V_G\} = \{\kappa'_n(v') \mid v' \in V_{G'}\}$, then R' is also accepting.

Proof. It is easy to see that $G_{\kappa_0} = G_{\sigma\circ\lambda}$ implies $G'_{\kappa'_0} = G'_{\sigma\circ\lambda'}$, and $G_{\kappa_{i+1}} \in \delta^{\circledast}(G_{\kappa_i})$ implies $G'_{\kappa'_{i+1}} \in \delta^{\circledast}(G'_{\kappa'_i})$, for $0 \leq i \leq n-1$, and $G_{\kappa_n} \in Q_{\mathrm{P}}^{G}$ implies $G'_{\kappa'_n} \in Q_{\mathrm{P}}^{G'}$. Thus, the sequence R' is a legal run of $\mathcal{A}$ on $G'_{\lambda'}$. ■

Next, we want to show that NDGAs are, to a certain extent, blind to symmetry. To this end, we define a mirroring operation, which introduces symmetry into a (labeled) graph by duplicating a given subgraph, together with its connections to the rest of the graph.

5.1.4 Definition (Graph Mirroring).
Consider a labeled graph $G_\lambda \in \Sigma^{\circledast}$ and a subset of nodes $U \subseteq V_G$. Further, let U' be a copy of U that is disjoint from V_G, and let $f \colon U \to U'$ be some bijection. The graph obtained by mirroring U in G_λ, denoted $\mathrm{mir}(G_\lambda, U)$, is defined as the labeled graph $G'_{\lambda'}$, such that

- $V_{G'} = V_G \cup U'$,
- $$\begin{aligned} \xrightarrow{\gamma}_{G'} = \xrightarrow{\gamma}_G &\cup \big\{ \langle u, f(v) \rangle \bigm| u \in (V_G \setminus U) \ \wedge \ v \in U \ \wedge \ u \xrightarrow{\gamma}_G v \big\} \\ &\cup \big\{ \langle f(u), v \rangle \bigm| u \in U \ \wedge \ v \in (V_G \setminus U) \ \wedge \ u \xrightarrow{\gamma}_G v \big\} \\ &\cup \big\{ \langle f(u), f(v) \rangle \bigm| u, v \in U \ \wedge \ u \xrightarrow{\gamma}_G v \big\}, \end{aligned}$$
  for every $\gamma \in \Gamma$,
- $\lambda'(v) = \lambda(v)$ for every $v \in V_G$, and $\lambda'(f(v)) = \lambda(v)$ for every $v \in U$.

We call f a mirroring bijection between U and U' in $G'_{\lambda'}$, and $f(v)$ a mirror image of v in $G'_{\lambda'}$, for every node $v \in U$.

Note that graph mirroring is well-defined because we consider graphs only up to isomorphism.

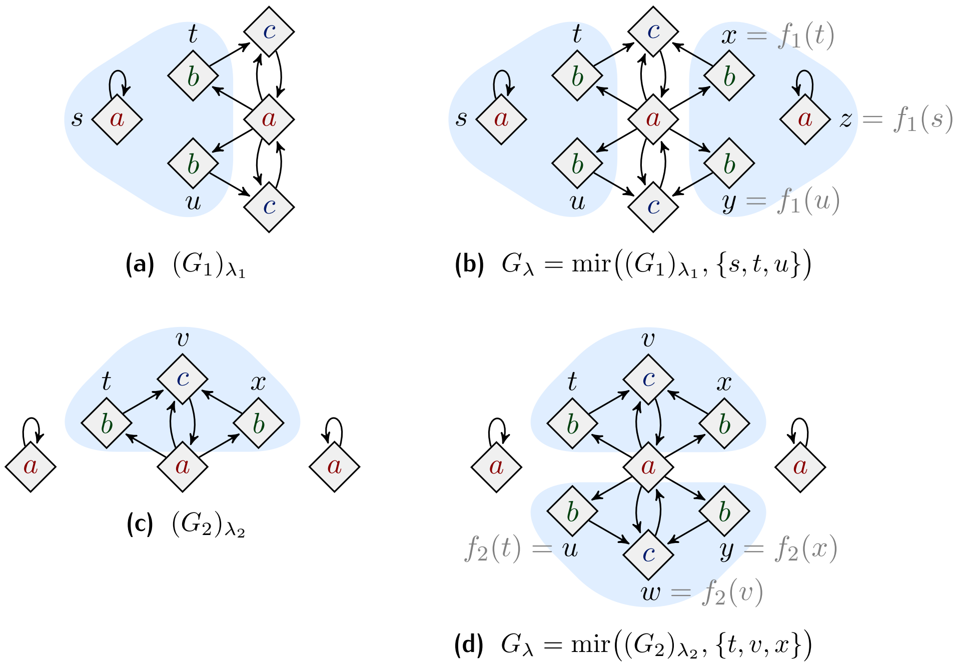


Figure 5.1. Mirroring in $\{a, b, c\}$-labeled $\{\square\}$-graphs.

5.1.5 Example.
Let $\Sigma = \{a, b, c\}$ and $\Gamma = \{\square\}$. Mirroring $\{s, t, u\}$ in the graph $(G_1)_{\lambda_1}$ from Fig. 5.1a yields the graph G_λ depicted in Fig. 5.1b. The function f_1 indicated in that figure is the only possible mirroring bijection between $\{s, t, u\}$ and $\{x, y, z\}$ in G_λ. We can obtain the same graph G_λ by mirroring $\{t, v, x\}$ in the graph $(G_2)_{\lambda_2}$ from Fig. 5.1c, as shown in Fig. 5.1d. The function f_2 is one of two possible mirroring bijections between $\{t, v, x\}$ and $\{u, w, y\}$ in G_λ (for the other one, the images of t and x are swapped). Since there are several ways of obtaining G_λ through mirroring, some of the nodes have several mirror images in G_λ. For instance, t has three of them: u, x and y.

We can now use the notion of graph mirroring to establish a necessary condition for NDGA-recognizability.

5.1.6 Lemma (Mirroring Lemma).
Every NDGA-recognizable graph language L is closed under mirroring, i.e., for every labeled graph G_λ and subset of nodes $U \subseteq V_G$,

$$G_\lambda \in L \quad \text{implies} \quad \mathrm{mir}(G_\lambda, U) \in L.$$

Proof. Let $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be an NDGA. Consider any $G_\lambda \in \Sigma^{\textcircled{\scriptsize$\Gamma$}}$ and $U \subseteq V_G$. We set $G'_{\lambda'} = \mathrm{mir}(G_\lambda, U)$, and fix a mirroring bijection $f \colon U \to (V_{G'} \setminus V_G)$ in $G'_{\lambda'}$. If $G_\lambda \in \mathrm{L}(\mathcal{A})$, then there must be an accepting run $R = G_{\kappa_0} \cdots G_{\kappa_n}$ of $\mathcal{A}$ on G_λ. By Remark 5.1.3, we can derive from it an accepting run $R' = G'_{\kappa'_0} \cdots G'_{\kappa'_n}$ of $\mathcal{A}$ on $G'_{\lambda'}$, in which the behaviour of every node $v \in V_G$ remains the same, i.e., $\kappa'_i(v) = \kappa_i(v)$, for $0 \le i \le n$, and every node $v \in U$ is imitated by its mirror image under f, i.e., $\kappa'_i(f(v)) = \kappa_i(v)$, for $0 \leq i \leq n$. (The local view of the nodes in $(V_G \setminus U)$ does not change because they cannot distinguish between nodes that are in the same state.) Consequently, $G'_{\lambda'} \in \mathrm{L}(\mathcal{A})$. ∎

Lemma 5.1.6 directly yields the following corollary.

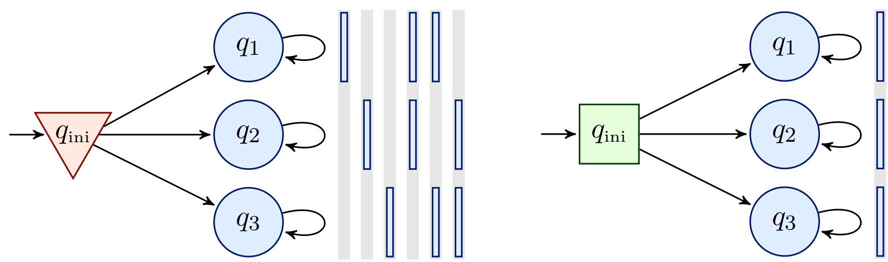
5.1.7 Corollary.
Every nonempty NDGA-recognizable graph language is (countably) infinite.

This implies that we have lost some expressive power by forbidding universal branchings.

5.1.8 Lemma ($\mathcal{L}_{\mathrm{NDGA}} \subset \mathcal{L}_{\mathrm{ADGA}}$).
There are (infinitely many) ADGA-recognizable graph languages that are not NDGA-recognizable.

Proof. Let $\Sigma = \Gamma = \{\square\}$. We consider the language $L_2^{\max}$ of all graphs that have at most two nodes, i.e., $L_2^{\max} = \{G \in \Sigma^{\circledR} \mid |V_G| \leq 2\}$. This language is nonempty and finite, which by Corollary 5.1.7 implies that it is not NDGA-recognizable. On the other hand, it is clearly ADGA-recognizable, since it is recognized by the ADGA $\mathcal{A}_2^{\max}$, specified in Fig. 5.2a. (The accepting configurations of that automaton are those that comprise at most two different states.)

We can apply the same reasoning to any graph language $L_c^{\max} \subseteq \Sigma^{\circledR}$ in which the number of nodes is limited by a constant $c \geq 1$. ∎



(a) $\mathcal{A}_2^{\max}$ **(b)** $\mathcal{A}_3^{\min} = \overline{\mathcal{A}}_2^{\max}$

Figure 5.2. $\mathcal{A}_2^{\max}$ and $\mathcal{A}_3^{\min}$, two ADGAs over $\langle\{\square\},\{\square\}\rangle$ whose graph languages consist of the graphs that have at most two nodes, and at least three nodes, respectively.

Following this line of thought, we also get that NDGAs cannot, in general, be complemented.

5.1.9 Lemma (Complementation).
The class $\mathcal{L}_{\mathrm{NDGA}}$ of NDGA-recognizable graph languages is *not* closed under complementation.

Proof. As mentioned in the proof of Lemma 5.1.8, the language $L_2^{\max}$ of all $\{\square\}$-labeled $\{\square\}$-graphs that have at most two nodes is not NDGA-recognizable. However, its complement, the language $L_3^{\min}$ of all graphs that have at least three nodes, is recognized by the NDGA $\mathcal{A}_3^{\min}$ specified in Fig. 5.2b. ∎

However, all the other closure properties of ADGAs mentioned in Section 3.6 are preserved.

5.1.10 Lemma (Closure Properties).
The class $\mathcal{L}_{\mathrm{NDGA}}$ of NDGA-recognizable graph languages is effectively closed under union, intersection and projection.

Proof. The union construction from Lemma 3.6.4 and the projection construction from Lemma 3.6.6 do not introduce any universal states, and thus yield NDGAs when applied on NDGAs.(1)

It remains to show closure under intersection. This can be done using a simple product construction, similar to the one for finite automata on words. Consider two NDGAs $\mathcal{A}_1 = \langle \Sigma, \Gamma, \widehat{Q}_1, \sigma_1, \delta_1, \mathcal{F}_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, \Gamma, \widehat{Q}_2, \sigma_2, \delta_2, \mathcal{F}_2 \rangle$. Without loss of generality, we may assume that they share the same node and edge alphabets, and that they are both nonblocking (see Remark 3.4.2). For any set $S \subseteq Q_1 \times Q_2$, we define the projection on the first component as

$$\mathrm{prj}_1(S) \coloneqq \big\{ q_1 \in Q_1 \;\big|\; \exists q_2 \in Q_2 \colon \langle q_1, q_2 \rangle \in S \big\},$$

and analogously for the projection $\mathrm{prj}_2(S)$ on the second component. With this, we construct the product NDGA $\mathcal{A}_{\otimes} = \langle \Sigma, \Gamma, \widehat{Q}_{\otimes}, \sigma_{\otimes}, \delta_{\otimes}, \mathcal{F}_{\otimes} \rangle$, where

- $(Q_{\otimes})_{\mathrm{P}} = (Q_1)_{\mathrm{P}} \times (Q_2)_{\mathrm{P}}, \quad (Q_{\otimes})_{\exists} = \big(Q_1 \times Q_2\big) \setminus (Q_{\otimes})_{\mathrm{P}},$
- $\sigma_{\otimes}(a) \ = \big\langle \sigma_1(a), \sigma_2(a) \big\rangle, \quad$ for every $a \in \Sigma$,
- $\delta_{\otimes}\big(\langle q_1, q_2 \rangle, \widehat{S}\big) = \delta_1\big(q_1, \big\langle \mathrm{prj}_1(S_\gamma) \big\rangle_{\gamma \in \Gamma}\big) \times \delta_2\big(q_2, \big\langle \mathrm{prj}_2(S_\gamma) \big\rangle_{\gamma \in \Gamma}\big),$
  for every $\langle q_1, q_2 \rangle \in Q_{\otimes}$ and $\widehat{S} = \langle S_\gamma \rangle_{\gamma \in \Gamma} \in (2^{Q_{\otimes}})^{\Gamma}$,
- $\mathcal{F}_{\otimes} = \big\{ F \subseteq (Q_{\otimes})_{\mathrm{P}} \;\big|\; \mathrm{prj}_1(F) \in \mathcal{F}_1 \,\wedge\, \mathrm{prj}_2(F) \in \mathcal{F}_2 \big\}.$

It is easy to see that $\mathcal{A}_{\otimes}$ recognizes $\mathrm{L}(\mathcal{A}_1) \cap \mathrm{L}(\mathcal{A}_2)$.

Additionally, we also get an alternative construction for the union by changing the definition of $\mathcal{F}_{\otimes}$ to

$$\mathcal{F}_{\otimes} = \big\{ F \subseteq (Q_{\otimes})_{\mathrm{P}} \;\big|\; \mathrm{prj}_1(F) \in \mathcal{F}_1 \,\vee\, \mathrm{prj}_2(F) \in \mathcal{F}_2 \big\}.$$

While this is significantly less efficient than the union construction from Lemma 3.6.4, in terms of number of states, it has the advantage of not relying on nondeterminism. It thus remains applicable when we restrict ourselves to deterministic automata in the next section. ■

As we have already seen in the Mirroring Lemma (Lemma 5.1.6), all NDGAs have some runs containing redundancies that prevent them from distinguishing between some nodes of the input graph. In fact, we can show that *every* run

(1) Note that those constructions require the input NDGAs to satisfy certain properties. Fortunately, they can be assumed to hold without loss of generality: The constructions for nonblocking and trim ADGAs (see Remarks 3.4.2 and 3.4.4) remain valid when restricted to NDGAs. Furthermore, the requirement that two automata agree on the sequence of quantifiers is trivially fulfilled for NDGAs.

on a sufficiently large input graph will contain such redundancies. In order to formally express this idea, we first define two node merging operations.

Given a graph and two of its nodes w and w', asymmetrically merging w' into w means to remove w', together with its adjacent edges, and add outgoing edges from w to all of the former *outgoing* neighbors of w'. If we additionally replicate the incoming edges of w', the merging is called symmetric.

5.1.11 Definition (Node Merging).
Consider a labeled graph $G_\lambda \in \Sigma^{\textcircled{\scriptsize\Gamma}}$ and two nodes $w, w' \in V_G$. We say that a labeled graph $G'_{\lambda'}$ is obtained by asymmetric merging of w' into w in G_λ, and denote it by $\mathrm{amrg}(G_\lambda, w, w')$, if

- $V_{G'} \;= V_G \setminus \{w'\}$,
- $\overset{\gamma}{\rightharpoonup}_{G'} \;= \big(\overset{\gamma}{\rightharpoonup}_G \cap (V_{G'} \times V_{G'})\big) \cup \big\{\langle w, v\rangle \bigm| w' \overset{\gamma}{\rightharpoonup}_G v\big\}$, for every $\gamma \in \Gamma$,
- $\lambda'(v) = \lambda(v)$, for every $v \in V_{G'}$.

Similarly, if instead of the second condition it holds that

$$\overset{\gamma}{\rightharpoonup}_{G'} = \big(\overset{\gamma}{\rightharpoonup}_G \cap (V_{G'} \times V_{G'})\big) \cup \big\{\langle w, v\rangle \bigm| w' \overset{\gamma}{\rightharpoonup}_G v\big\} \cup \big\{\langle v, w\rangle \bigm| v \overset{\gamma}{\rightharpoonup}_G w'\big\},$$

for every $\gamma \in \Gamma$, then $G'_{\lambda'}$ is said to be obtained by symmetric merging of w and w' in G_λ, and is denoted by $\mathrm{smrg}(G_\lambda, w, w')$.

With the notation defined, we can now derive another necessary condition for NDGA-recognizability. The following result reminds strongly of the Pumping Lemma for regular word languages, and its proof is somewhat similar in spirit (also based on the Pigeonhole Principle).

5.1.12 Lemma (Merging Lemma).
For every NDGA-recognizable graph language L there exist natural numbers n and m (with $n < m$) such that every labeled graph G_λ satisfies the following node merging properties:

- If G_λ has at least n nodes, then there exist nodes $w, w' \in V_G$ such that
  $$G_\lambda \in L \quad \text{implies} \quad \mathrm{amrg}(G_\lambda, w, w') \in L.$$
- If G_λ has at least m nodes, then there exist nodes $w, w' \in V_G$ such that
  $$G_\lambda \in L \quad \text{implies} \quad \mathrm{smrg}(G_\lambda, w, w') \in L.$$

Moreover, if $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$ is an NDGA that recognizes L, then we have $n \le |Q|^{\mathrm{len}(\mathcal{A})+1}$, and $m \le \big(|Q| \cdot 2^{|\Gamma|\cdot|Q|}\big)^{\mathrm{len}(\mathcal{A})+1}$.

Proof. We fix an NDGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$ that recognizes L and use the abbreviations $g = |\Gamma|$, $s = |Q|$, and $\ell = \mathrm{len}(\mathcal{A})$.

Note that there cannot be more than $s^{\ell+1}$ different local sequences of states in a run of $\mathcal{A}$. Consider a labeled graph $G_\lambda \in \Sigma^{\textcircled{\scriptsize\Gamma}}$ that has more than $s^{\ell+1}$ nodes, and some run $R = G_{\kappa_0} \cdots G_{\kappa_m}$ of $\mathcal{A}$ on G_λ, where $m \le \ell$. By the Pigeonhole Principle, there must be two distinct nodes $w, w' \in V_G$ that have the same local sequence of states in R. If we asymmetrically merge w' into w, i.e., if we construct $G'_{\lambda'} = \mathrm{amrg}(G_\lambda, w, w')$, the remaining nodes in $G'_{\lambda'}$ will not see the difference if they maintain their behaviour from R, since their local views will remain the same. More formally, by Remark 5.1.3, we can derive from R a run $R' = G'_{\kappa'_0} \cdots G'_{\kappa'_m}$ of $\mathcal{A}$ on $G'_{\lambda'}$, such that $\kappa'_i(v) = \kappa_i(v)$ for every node $v \in V_{G'}$ and $0 \leq i \leq m$. If R is accepting, so is R', hence $G_\lambda \in \mathrm{L}(\mathcal{A})$ implies $G'_{\lambda'} \in \mathrm{L}(\mathcal{A})$.

If we want to symmetrically merge two nodes, the reasoning is very similar, but slightly more involved because the merged node inherits the unified incoming neighborhood of the original nodes, and consequently would get a new local view if the two local views of the original nodes were different. A simple solution is to require a larger minimum number of nodes. Altogether, there cannot be more than $(s\,2^{gs})^{\ell+1}$ different local views in any run of $\mathcal{A}$. Again by the Pigeonhole Principle, if a graph has more than $(s\,2^{gs})^{\ell+1}$ nodes, there must be two distinct nodes that have the same local view. The rest of the argument is analogous to the previous scenario. ■

While the previously seen Mirroring Lemma (Lemma 5.1.6) allows us to enlarge ("pump up") graphs without leaving a given NDGA-recognizable graph language, the Merging Lemma allows us to shrink ("pump down") some of them. The combination of both could thus be considered as some sort of "graph pumping lemma".

The Merging Lemma provides further evidence of the expressive weakness of NDGAs, but, perhaps more importantly, it also tells us that their emptiness problem is decidable. The daunting time complexities indicated in the following lemma are only rough upper bounds. Better estimates and algorithms can hopefully be found through further investigation.

5.1.13 Lemma (Emptiness Problem).
The emptiness problem of NDGAs is decidable in doubly-exponential time. More precisely, for every NDGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$,

- whether its recognized graph language $\mathrm{L}(\mathcal{A})$ is empty or not can be decided in time 2^k, where $k \in \mathrm{O}\big(|\Gamma| \cdot |Q|^{4\,\mathrm{len}(\mathcal{A})} \cdot \mathrm{len}(\mathcal{A})\big)$, and
- whether its undirected graph language $\mathrm{L}_{\updownarrow}(\mathcal{A})$ is empty or not can be decided in time $2^{2^{k'}}$, where $k' \in \mathrm{O}\big(|\Gamma| \cdot |Q| \cdot \mathrm{len}(\mathcal{A})\big)$.

Proof. We use again the abbreviations $g = |\Gamma|$, $s = |Q|$, and $\ell = \mathrm{len}(\mathcal{A})$. By applying the first part of the Merging Lemma (Lemma 5.1.12) recursively, we conclude that if $\mathrm{L}(\mathcal{A})$ is not empty, then it contains a labeled graph that has at most $s^{\ell+1}$ nodes. Similarly, by the second part of the Merging Lemma, if $\mathrm{L}_{\updownarrow}(\mathcal{A})$ is not empty, then it contains an undirected labeled graph that has at most $(s\,2^{gs})^{\ell+1}$ nodes. (For undirected graphs, asymmetric merging is, in general, not applicable.) Hence, the emptiness problem is decidable because the search space is finite.

We now derive a rough asymptotic upper bound on the time complexities of the naive approaches that check every (directed) graph that has at most $s^{\ell+1}$ nodes, and every undirected graph that has at most $(s\,2^{gs})^{\ell+1}$ nodes, respectively.

- The maximum numbers of nodes can be over-approximated by $\mathrm{O}(s^{2\ell})$ and $\mathrm{O}(2^{4gs\ell})$, respectively.
- Given a natural number n, there are $\mathrm{O}(2^{gn^2})$ $\mathit{\Gamma}$-graphs with precisely n nodes. (This is only an upper bound because we consider isomorphic graphs to be equal.)
- Given a $\mathit{\Gamma}$-graph G with n nodes, we can decide whether $G_\lambda \in \mathrm{L}(\mathcal{A})$ for some labeling $\lambda\colon V_G \to \mathit{\Sigma}$, by checking every possible run of $\mathcal{A}$ that starts with a configuration $G_{\kappa_0} \in \big(\sigma(\mathit{\Sigma})\big)^G$. This can be done in time $\mathrm{O}(s^{n(\ell+1)}) \subseteq \mathrm{O}(s^{2n\ell})$.

Hence, the total time complexities are bounded by

$$\mathrm{O}\left(\sum_{n=1}^{s^{2\ell}} 2^{gn^2} \cdot s^{2n\ell}\right) \subseteq \mathrm{O}\left(2^{8gs^{4\ell}\ell}\right), \quad \text{and}$$

$$\mathrm{O}\left(\sum_{n=1}^{2^{4gs\ell}} 2^{gn^2} \cdot s^{2n\ell}\right) \subseteq \mathrm{O}\left(2^{2^{16gs\ell}}\right),$$

respectively. ■

5.2 Deterministic Distributed Graph Automata

As a further restriction, we now forbid nondeterministic choices.

5.2.1 Definition (Deterministic Distributed Graph Automaton).
A deterministic distributed graph automaton (DDGA) is a *nonblocking* NDGA $\mathcal{A} = \langle \mathit{\Sigma}, \mathit{\Gamma}, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ in which every state $q \in Q$ has at most one outgoing transition for every $\widehat{S} \in (2^Q)^{\mathit{\Gamma}}$, i.e., $|\delta(q, \widehat{S})| \leq 1$. We denote by $\mathcal{L}_{\mathrm{DDGA}}$ the class of all DDGA-recognizable graph languages.

The transition function being deterministic forces every node of an input graph to behave like its mirror images. This allows us to state a stronger Mirroring Lemma for DDGAs.

5.2.2 Lemma (Strong Mirroring Lemma).
Every DDGA-recognizable graph language L is closed under both mirroring and the reversal of mirroring, i.e., for every labeled graph G_λ and subset of nodes $U \subseteq V_G$,

$$G_\lambda \in L \quad \text{if and only if} \quad \mathrm{mir}(G_\lambda, U) \in L.$$

Proof.

($\Rightarrow$) The "only if" direction is a specialization of the weaker Mirroring Lemma (Lemma 5.1.6) to DDGAs.

($\Leftarrow$) Let $\mathcal{A} = \langle \mathit{\Sigma}, \mathit{\Gamma}, \widehat{Q}, \sigma, \delta, \mathcal{F} \rangle$ be a DDGA. Consider any $G_\lambda \in \mathit{\Sigma}^{\textcircled{\scriptsize\Gamma}}$ and $U \subseteq V_G$. We set $G'_{\lambda'} = \mathrm{mir}(G_\lambda, U)$, and fix a mirroring bijection $f\colon U \to (V_{G'} \setminus V_G)$ in $G'_{\lambda'}$. If $G'_{\lambda'} \in \mathrm{L}(\mathcal{A})$, then the (unique) run $R' = G'_{\kappa'_0} \cdots G'_{\kappa'_n}$

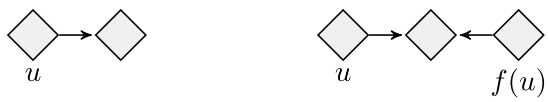
of $\mathcal{A}$ on $G'_{\lambda'}$ is accepting. Since $\mathcal{A}$ is deterministic, it can be shown inductively that every node $v \in U$ behaves like its mirror image $f(v)$, i.e., $\kappa'_i(v) = \kappa'_i(f(v))$, for $0 \leq i \leq n$. If we remove $f(v)$ for every $v \in U$, we do not change the local view of any node in V_G. Hence, by Remark 5.1.3, we can derive from R' the accepting run $R = G_{\kappa_0} \cdots G_{\kappa_n}$ of $\mathcal{A}$ on G_λ, where $\kappa_i(v) = \kappa'_i(v)$, for every $v \in V_G$ and $0 \leq i \leq n$. Consequently, $G_\lambda \in \mathrm{L}(\mathcal{A})$. ■

The Strong Mirroring Lemma implies that DDGAs are utterly incapable of breaking symmetry. This makes them strictly weaker than NDGAs.

5.2.3 Lemma ($\mathcal{L}_{\mathrm{DDGA}} \subset \mathcal{L}_{\mathrm{NDGA}}$).
There are (infinitely many) NDGA-recognizable graph languages that are not DDGA-recognizable.

Proof. Let $\varSigma = \varGamma = \{\square\}$. We consider again the language $L_3^{\min} = \{G \in \varSigma^{\textcircled{\varGamma}} \mid |V_G| \geq 3\}$ of all graphs that have at least three nodes. As already mentioned in the proof of Lemma 5.1.9, this language is NDGA-recognizable because it is recognized by the NDGA $\mathcal{A}_3^{\min}$ from Fig. 5.2b. Now, assume that $L_3^{\min}$ is also DDGA-recognizable and consider the graphs G and $G' = \mathrm{mir}(G, \{u\})$ from Fig. 5.3. Clearly $G' \in L_3^{\min}$, and by Lemma 5.2.2, it follows that $G \in L_3^{\min}$, which is a contradiction. Hence, $L_3^{\min}$ is not DDGA-recognizable.

We can apply a similar reasoning to any graph language $L_c^{\min} \subseteq \varSigma^{\textcircled{\varGamma}}$ in which the number of nodes is bounded from below by a constant $c \geq 2$. ■



(a) G **(b)** $G' = \mathrm{mir}(G, \{u\})$

Figure 5.3. Two $\{\square\}$-labeled $\{\square\}$-graphs related by mirroring.

Instead of expecting the automaton to break symmetry through nondeterminism, we could also rely on additional information provided by the node labels of the input graph. Well-chosen labels can make the structure of the input graph visible to a deterministic automaton. This strong dependence on the node labeling implies the following negative result.

5.2.4 Lemma (Projection).
The class $\mathcal{L}_{\mathrm{DDGA}}$ of DDGA-recognizable graph languages is *not* closed under projection.

Proof. Let $\varSigma = \{a, b, c\}$ and $\varGamma = \{\square\}$. We consider the language $L_{a,b,c}^{\mathrm{occur}}$ of all $\varSigma$-labeled graphs in which every node label occurs at least once, i.e.,

$$L_{a,b,c}^{\mathrm{occur}} = \{G_\lambda \in \varSigma^{\textcircled{\varGamma}} \mid \exists u, v, w \in V_G \colon \lambda(u) = a \wedge \lambda(v) = b \wedge \lambda(w) = c\}.$$

This language is recognized by the DDGA $\mathcal{A}_{a,b,c}^{\mathrm{occur}}$ specified in Fig. 5.4. Now, consider the projection $h \colon \varSigma \to \{\square\}$ with $h(a) = h(b) = h(c) = \square$. Clearly, $h(L_{a,b,c}^{\mathrm{occur}})$ is equal to the language $L_3^{\min}$ of all $\{\square\}$-labeled graphs that have at least three nodes. As shown in the proof of Lemma 5.2.3, this language is not DDGA-recognizable. ∎

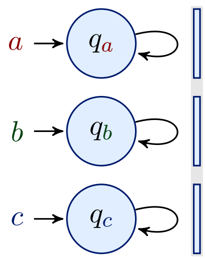


Figure 5.4. $\mathcal{A}^{\text{occur}}_{a,b,c}$, a DDGA over $\langle\{a,b,c\},\{\square\}\rangle$ whose graph language consists of the labeled graphs in which each of the three node labels occurs at least once.

On the positive side, the union and intersection constructions for NDGAs remain valid, and complementation becomes trivial.

5.2.5 Lemma (Closure Properties).
The class $\mathcal{L}_{\text{DDGA}}$ of DDGA-recognizable graph languages is effectively closed under boolean set operations.

Proof. The product constructions for union and intersection specified in the proof of Lemma 5.1.10 yield DDGAs when applied on DDGAs.

It remains to show closure under complementation. Consider any DDGA $\mathcal{A} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, \mathcal{F}\rangle$. Since its transition function is deterministic, $\mathcal{A}$ has precisely one run on each labeled graph $G_\lambda \in \Sigma^{\textcircled{\Gamma}}$. Hence, we obtain a complement automaton $\bar{\mathcal{A}}$ by simply complementing the acceptance condition, i.e., $\bar{\mathcal{A}} = \langle \Sigma, \Gamma, \widehat{Q}, \sigma, \delta, 2^{Q_P} \setminus \mathcal{F}\rangle$. ∎

Chapter 6

Conclusion

We first summarize and comment upon the results obtained in this work, and then conclude with a small selection of open questions that seem worth pursuing.

6.1 Commented Summary

We have introduced ADGAs, a new class of finite graph automata. Although many graph automaton models have been defined over the last decades, ADGAs are probably the first automata to be equivalent to MSO-logic on graphs. In this regard, it seems remarkable that the individual ingredients of this model of computation are not spectacular at all, and, for the most part, well-established.

To a certain extent, ADGAs can be considered as synchronous distributed algorithms, where each node is limited to a finite-state machine. The model is further restricted by a constant running time and the fact that nodes see only an abstract representation of their neighborhood, in the form of sets of states, which drastically limits their ability to distinguish between different neighbors. The latter restriction allows ADGAs to operate on graphs of unbounded degree, a feature that sets them apart from many other types of finite graph automata defined in the literature (for instance in [WR79] and [Tho91]).

Besides this distributed character, there is also a centralized aspect to ADGAs, which contributes greatly to their expressive power. On the one hand, acceptance is decided on a global level, based on the set of states reached by the local processors. This combines and generalizes two decision-making approaches that may appear more natural in a distributed setting: decision by a unique leader and decision by unanimous agreement of all the nodes. On the other hand, ADGAs implement the powerful concept of alternation, a kind of parallelization, which is also expressed in terms of the global configuration of the entire system.

As already mentioned above, taken in isolation these concepts represent nothing new. The contribution of the present thesis is mainly to combine them into a model of computation that balances between distribution and centralization in a way that matches precisely MSO-logic.

For finite automata on words and (bottom-up) tree automata, alternation and nondeterminism do not increase expressiveness (see [CKS81, Thm 5.2] and [TATA08, Thm 7.4.1]). For our graph automata, on the other hand, they are essential ingredients that cannot be eliminated without losing expressive power.

We have seen that the deterministic, nondeterministic and alternating variants of distributed graph automata form a strict hierarchy, i.e.,

$$\mathcal{L}_{\text{DDGA}} \subset \mathcal{L}_{\text{NDGA}} \subset \mathcal{L}_{\text{ADGA}}.$$

On an intuitive level, this is not very surprising, since nondeterministic choices and universal branchings are closely related to existential and universal quantification in MSO-logic, and removing one type of quantifier (without allowing to negate the other) drastically diminishes expressiveness.

Another way to look at this is from the perspective of the closure properties of the three variants of automata, which are summarized in Table 6.1. As already mentioned in-between the two proofs of Section 4.2, Engelfriet has characterized the class of MSO-definable graph languages as the smallest class that contains certain elementary graph languages and is closed under boolean set operations and under projection. To achieve closure under projection with our distributed automata, we need nondeterminism so that the nodes can guess which label they might have had without application of the projection function. But if the additional expressive power introduced by nondeterministic choices is not matched by the corresponding dual, namely universal branchings, then there is an asymmetry that makes us lose closure under complementation.

	Closure Properties				Decidability
	Complement	Union	Intersection	Projection	Emptiness
ADGA	✓	✓	✓	✓	✗
NDGA	✗	✓	✓	✓	✓
DDGA	✓	✓	✓	✗	✓

Table 6.1. Closure and decidability properties of alternating, nondeterministic, and deterministic distributed graph automata.

However, as also indicated in Table 6.1, if we do not enable universal branchings, it has the positive effect that emptiness remains decidable, which is not possible anymore once we have reached the expressive power of MSO-logic on graphs. The reason for this positive decidability result is that, for NDGAs, any run on a sufficiently large input graph contains redundancies. This allows us to narrow down the search space to a finite number of graphs.

6.2 Open Questions

In the present work, we have not gone much further than introducing new definitions. Whether these definitions are sensible largely depends on the insights that we might gain through them. In this regard, it would be very interesting to obtain answers to the following questions.

Logics Equivalent to NDGAs and DDGAs.

Although we have been mostly concerned with ADGAs here, the fact that emptiness is decidable for NDGAs and DDGAs might make their corresponding classes of graph languages attractive, despite their less robust closure properties. As mentioned in Section 4.3, Trakhtenbrot's

Theorem states that, even when restricting formulas to first-order logic, satisfiability is undecidable on graphs. This tells us that alternation in our graph automata is required to even cover the expressiveness of first-order logic, but also arouses curiosity regarding the logical equivalent of the weaker variants. *What would be logical formalisms on graphs that precisely define the NDGA- and DDGA-recognizable graph languages?*

Alternative Definitions of ADGAs.

The definition of ADGAs given in this thesis (Definition 3.2.1) seems quite involved, and to loosely paraphrase a famous French aviator, as long as there remains something nonessential to remove, there is also room for improvement. On some classes of graphs, the necessary running time of ADGAs can be bounded by a fixed constant. For instance, if we only consider (graphs representing) words, simulating the classical finite automata shows us that every regular language is recognizable by some ADGA of length 2. (Nondeterministically guess a run of the word automaton in the first round, then check that it is legal and accepting in the second round.) But on general graphs, there is no such fixed constant, as can be easily inferred from the infinity of the MSO quantifier alternation hierarchy investigated by Matz and Thomas in [MT97]. The question remains: *can we impose simplifications or restrictions on the definition of ADGAs without sacrificing expressive power?*

Impact on other Research.

Since ADGAs arose from an open-ended question, it seems only fitting to conclude this thesis with other questions of that type.

- On words and trees, the equivalence between finite automata and MSO-logic led to the decidability of the satisfiability and validity problems of MSO-logic. Unfortunately, this is not extendable to graphs. But maybe ADGAs can help us finding necessary conditions for MSO-definability, similar to what we got for NDGA-recognizability in the Mirroring Lemma (Lemma 5.1.6). More generally, we might ask: *what can ADGAs tell us about the class of MSO-definable graph languages?*
- Conversely, we could also use MSO-logic as a means to an end. As already mentioned, ADGAs can be considered, to some extent, as distributed algorithms. *What can the connection to MSO-logic tell us about distributed algorithms?*

Bibliography

[Büc60] J.R. Büchi (1960)
Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6, pages 66–92.

[CE12] B. Courcelle, J. Engelfriet (2012)
Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach. Cambridge University Press.

[CKS81] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer (1981)
Alternation. Journal of the ACM, 28, pages 114–133.

[Cou90] B. Courcelle (1990)
The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and computation 85, pages 12–75.

[Cou97] B. Courcelle (1997)
The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In Handbook of Graph Grammars, Volume 1: Foundations. G. Rozenberg ed., pages 313–400, World Scientific.

[Cou08] B. Courcelle (2008)
Graph structure and monadic second-order logic: Language theoretical aspects. In Automata, Languages and Programming, pages 1–13, Springer.

[Die10] R. Diestel (2010)
Graph Theory. Fourth Edition, Springer.

[Don70] J. Doner (1970)
Tree acceptors and some of their applications. Journal of Computer and System Sciences 4, pages 406–451.

[Elg61] C.C. Elgot (1961)
Decision problems of finite automata design and related arithmetics, Transactions of the American Mathematical Society 98, pages 21–51.

[Eng91] J. Engelfriet (1991)
A Regular Characterization of Graph Languages Definable in Monadic Second-Order Logic. Theoretical Computer Science 88, pages 139–150, Elsevier.

[GH82] Y. Gurevich, L. Harrington (1982)
Trees, Automata and Games. Proceedings of the fourteenth annual ACM symposium on Theory of computing, pages 60–65.

[HGG97-99] G. Rozenberg, H. Ehrig, G. Engels, H.J. Kreowski, U. Montanari, eds. (1997–1999)
Handbook of Graph Grammars and Computing by Graph Transformation. Volumes 1–3, World Scientific.

[Kum06] K.N. Kumar (2006)
Alternating Automata. Notes on Automata, Logics, Games and Algebra, Lecture 6. `http://www.cmi.ac.in/~kumar/words`

[Kur30] K. Kuratowski (1930)
Sur le problème des courbes gauches en topologie. Fundamenta Mathematicae 15, pages 271–283 (in French).

[Lib04] L. Libkin (2004)
Elements of Finite Model Theory. Springer.

[LT00] C. Löding, W. Thomas (2000)
Alternating Automata and Logics over Infinite Words. Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, pages 521–535, Springer.

[Mil75] D.L. Milgram (1975)
Web Automata. Information and Control 29, pages 162–184.

[MT97] O. Matz, W. Thomas (1997)
The Monadic Quantifier Alternation Hierarchy over Graphs is Infinite. In Proceedings of 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97), pages 236–244.

[SMR73] A.N. Shah, D.L. Milgram, A. Rosenfeld (1973)
Parallel Web Automata. Technical Report 231, University of Maryland.

[TATA08] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi (2008)
Tree Automata Techniques and Applications.
`http://tata.gforge.inria.fr` or
`http://www.grappa.univ-lille3.fr/tata`

[Tho91] W. Thomas (1991)
On Logics, Tilings, and Automata. In J.L. Albert, B. Monien, M. Rodríguez-Artalejo, eds., ICALP, volume 510 of Lecture Notes in Computer Science, pages 441–454, Springer.

[Tho96] W. Thomas (1996)
Languages, Automata, and Logic. Bericht 9607, Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel.
Also published in Handbook of Formal Languages, Volume 3: Beyond Words, G. Rozenberg and A. Salomaa, eds., pages 389–455, Springer 1997.

[Tho97] W. Thomas (1997)
Automata Theory on Trees and Partial Orders. TAPSOFT'97: Theory and Practice of Software Development, Lecture Notes in Computer Science Volume 1214, pages 20–38, Springer.

[Tra50] B.A. Trakhtenbrot (1950)
The Impossibility of an Algorithm for the Decidability Problem on Finite Classes. Doklady Akademii Nauk SSSR 70, pages 569–572 (in Russian).

[Tra61] B.A. Trakhtenbrot (1961)
Finite automata and the logic of monadic predicates. Doklady Akademii Nauk SSSR 140, pages 326–329 (in Russian).

[TW68] J.W. Thatcher, J.B. Wright (1968)
Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2, pages 57–82.

[Wag37] K. Wagner (1937)
Über eine Eigenschaft der ebenen Komplexe. Mathematische Annalen 114, pages 570–590 (in German).

[WR79] A. Wu, A. Rosenfeld (1979)
Cellular Graph Automata. I. Basic Concepts, Graph Property Measurement, Closure Properties. Information and Control 42, pages 305–329.