# The complexity of membership problems for finite recurrent systems and minimal triangulations

**Dissertation**

**Zur Erlangung des akademischen Grades**
***Doktor der Naturwissenschaften***
(Dr.rer.nat.)

**Bayerische Julius-Maximilians-Universität Würzburg**
**Fakultät für Mathematik und Informatik**

von

Daniel Meister

aus Jena

Würzburg, im Jahre 2006

# *Vorwort*

*Im Jahre 1994, es neigte sich dem Ende zu, und ich war auf dem Weg vom Krankenhaus zurück nach Hause, hörte ich es im Radio. Oder eigentlich hörte ich es nicht, sondern verstand nur ein paar Brocken, Fetzen Inhalts. Es ging um eine Reform der Rechtschreibung. Ich hätte schockiert sein müssen, war es auch, verwarf aber jeden weiteren Gedanken an das Thema unverzüglich, da einerseits im Straßenverkehr Obacht und Aufmerksamkeit oberste Gebote sind und andererseits ich zuvor nie von solcherlei Reformbestrebungen vernahm, so daß doch recht schnell klar sein mußte, daß ich mich* ver*hört hatte. Zeit verstrich, und das Thema trat nicht wieder in mein Bewußtsein. Vorerst!*

*Wie schlimm war es dann, als es eines Tages doch dazu kam, was aus heutiger Sicht natürlich unvermeidlich war: Die Kultusminister beschlossen eine Reform der Rechtschreibung, die dann irgendwie umgesetzt werden sollte. Ich kann gar nicht genau sagen, wie das Thema damals in der Öffentlichkeit behandelt wurde, ich war ja schließlich noch frischer Student und hatte mit Studium zu tun. Vielleicht war es auch so, daß erst einmal gar nichts passierte, da die neuen Regeln erst nach einer gewissen Übergangsfrist verbindlich wurden und vor allem nur für „öffentliche Einrichtungen“, wie da wären vor allem Schulen und Behörden.*

*Man kann darüber spekulieren, was überhaupt der Auslöser war, eine solche Reform anzustrengen. Ich hoffe, daß es nicht daran lag, daß die letzte Reform noch im Kaiserreich beschlossen wurde, denn so reformfreudig war man in Deutschland in den neunziger Jahren des vergangenen Jahrhunderts nicht. Worüber es aber keiner Spekulationen bedarf, sind die Auswirkungen dieser Reform. (Das Wort* Reform *kommt so häufig vor; tatsächlich bewirkt nicht jede Reform eine Wendung zum besseren.) Für mich persönlich kann ich sagen, daß sich meine Kenntnisse der Deutschen Schriftsprache stark verbessert haben, da ich beim Lesen eines Textes gezwungen war herauszufinden, welchen Regeln der jeweilige Autor folgte, um nun wiederum zu wissen, was genau er aussagen wollte. Dennoch war ich oft genug mit dem Problem konfrontiert, nicht genau zu wissen, ob es Absicht war oder Zufall, daß da stand, was da stand. Und dies ist wohl einer der größten Vorwürfe, den man Politikern machen kann: daß sie bewußt eine einstmals homogene und in weiten Teilen beherrschte Schriftsprache derart zerrüttet haben, daß kein normaler Nutzer heutzutage gegen Fehler gefeit ist, die vor nicht allzu langer Zeit ausgeschlossen waren. Ich verstehe Schrift und Schriftsprache als Kulturgüter höchster Bedeutung. Und ich denke, der Umgang mit ihnen zeugt von der Wertschätzung, die ihnen entgegengebracht wird. Aber darum soll es hier ja eigentlich gar nicht gehen.*

*Wieso habe ich aber das Thema Rechtschreibreform aufgegriffen und damit begonnen? Es ist ein sehr aktuelles Thema, und es ist ein Thema, welches mich seit zehn Jahren konstant beschäftigt. Ich hatte das Glück, um Erich Kästners Worte zu variieren, aufzuwachsen, (nicht dort, wo Milch und Honig von den Bäumen tropfen, sondern) wo Kunst und Kultur eine herausragende Rolle spielen, wo sie quasi zum* *persönlichen Selbstverständnis gehören. Natürlich meine ich das nicht pauschal, und die Leute liefen nicht durch die Gassen Goethe und Schiller rezitierend in einem fort. Dennoch trifft man pausenlos auf Kultur. Ich habe mir als Sinnbild das Paar* Kultur und Natur *gewählt, und genau das beschreibt, was mir besonders wichtig ist. Und natürlich gehört zum Bereich Kultur Schriftsprache, und natürlich muß diese hochentwickelt, differenzierend, ausdrucksfähig sein. Wie sieht es denn heute (gemäß der Reform) damit aus? – Wenn man einmal davon absieht, daß keiner die aktuellen Regeln kennt, da einfach viel zu häufig am Reformwerk herumreformiert wurde, kommt der Aspekt Differenzierbarkeit/Ausdrucksfähigkeit einfach zu kurz. Auch war es einmal so (und ist es noch im täglichen Gebrauch), daß sich die Schrift an der Sprache orientierte. Damit war Deutsch eine der wenigen Sprachen, die es erlaubten, vom geschriebenen Wort recht schnell auf Aussprache und Sinn zu schließen. Welchen Regeln soll man denn dann bei Wortungetümen wie „platzieren“ oder, für den theoretischen Informatiker ganz wichtig, „nummerieren“ folgen? Ganz zu schweigen von der semantischen Verbindung zwischen* platzen *und „platzieren“. Jedenfalls bin ich der Meinung, daß die Politik gerade hierbei bewiesen hat, daß ihr manche Themen, die offiziell ganz oben auf hochgehaltenen Fahnen stehen, von untergeordneter Bedeutung sind.*

*Ich könnte, die einen wissen es, die anderen können es sich lebhaft vorstellen, zu diesem Thema eine längere Abhandlung verfassen. Zum Glück nehmen mir andere diese Arbeit ab. Ich finde, ein wenig Bewahren kann auch nicht schaden. Aber das soll hier ja nicht nur stehen. Schließlich ist die vorliegende Arbeit dem Bereich Theoretische Informatik zuzuordnen, und eigentlich möchte der Leser an dieser Stelle wissen, wem der Autor –* ich *in dem Falle – dankt.*

*Es ist schwierig, dies in vollem Umfang zu sagen. Ich hatte das ungemeine Glück, viele, viele Jahre vor dem Abitur bereits zu wissen, daß ich unbedingt Informatik studieren wollte. Einen Rechner hatte ich da noch nicht, ich hatte auch noch nicht an einem gesessen. Ich kannte Rechner nur aus dem Fernsehen und war dennoch fasziniert. Es war also nicht geboten, nach Schulabschluß erst einmal einen Selbstfindungsprozeß zu initiieren, um herauszufinden, was ich denn eigentlich könne. Normalerweise sollte dann die bewußte Wahl der Universität getroffen werden. Ich ging ziemlich pragmatisch vor und wählte die Uni nach einem naheliegenden Kriterium. Ich konnte ja nicht wissen, daß die Auswahl ein absolutes Optimum wäre. Ich begann also an der Friedrich-Schiller-Universität. Und ich hatte das Glück, bei hervorragenden Dozenten Vorlesungen besuchen zu können, die mich im Laufe meines Studiums aktiv und passiv maßgeblich beeinflußten. Zu nennen sind Dieter Kratsch und Gerd Wechsung. Beide hielten die besten Vorlesungen meines gesamten Studiums, und da ist es ja auch kein Wunder, daß mich die entsprechenden Themen noch immer beschäftigen. Im übrigen bin ich froh, daß es die Theoretische Informatik gibt; andernfalls wäre ich bestimmt nicht so glücklich über meine Studienwahl gewesen. (Aber wahrscheinlich doch.)*

Kultur und Natur – *davon gab es reichlich, und ich genoß es. Nicht so intensiv und bewußt wie heute, aber dennoch. Und ich wage zu behaupten, daß dieses Umfeld,*

*was nicht zuletzt auch durch große Geschichte geprägt wurde, mich stark beeinflußte und in mir liegenden Anlagen zum Ausbruch verhalf. Zum Leidwesen so manch anderer.*

*Daß ich dann nach Würzburg kam, war weder eine logische Folge noch Ergebnis bewußter Planungen. Es war reiner Zufall. Und das war gut so. Am Lehrstuhl von Klaus Wagner ließ sich hervorragend Forschung betreiben; die vorliegende Arbeit legt davon Zeugnis ab. Ihm ist vor allem für das Gelingen zu danken. Aber auch meinen Kollegen, die noch da sind oder auch schon nicht mehr, gebührt Dank. Altgedient sind Christian und Elmar. Mit beiden hatte ich meinen ersten Konferenzbeitrag. Elmar mußte außerdem so manche „Diskussion“ „ertragen“. Bernhard muß besonders erwähnt werden, nicht wegen der Diskussionen, sondern weil er einen Beitrag zu dieser Arbeit geleistet hat. Seinen Fähigkeiten ist der schöne Beweis zu Lemma 6.10 zu verdanken. Stephen möchte ich hier erwähnen. Unsere Bekanntschaft geht auf ein Sommerfest zurück, erstreckte sich über eine Diplomarbeit und mündete im Nebenzimmer (wenn ich das einmal so salopp abreißen darf). Ich glaube, manchmal streiten wir heftig (selbstverständlich nur verbal) über harmlose Themen, aber es trägt zur Meinungsbildung stark bei.*

*Es gibt aber nicht nur Arbeit (und das meine ich im Ernst und nicht witzig, ironisch oder mit Hintergedanken). Besonders im privaten Umfeld sind Menschen – Freunde, Bekannte, Verwandte – zu nennen, die für das Gelingen von unschätzbarer Bedeutung waren und sind. Ich habe lange überlegt, aber ich nenne keine Namen. Einerseits ist mir immer wieder aufgefallen, daß der eine vergessen wurde oder nicht gebührend bewertschätzt wurde, andererseits … nun ja. Stellvertretend bedanke ich mich herzlichst bei allen, die sich angesprochen fühlen bei der Erwähnung von Jena-Weimar und Dresden. Kultur und Natur in ihrer schönsten Form. Doch noch: der Familie.*

Würzburg, 10.03.2006
(Ausbesserungen am 14.07.2006)

# Table of contents

# Part I

# Introduction and theory fundamentals

*The thesis is partitioned into three parts, and every part provides a frame for the contained chapters. While Parts II and III are dedicated to new results, Part I has a preparatory character. Compared to Parts II and III, it is also short; it consists of only three chapters.*

*Chapter 1 is a real introduction into the topic of the thesis. It is explained what we basically mean by the* complexity of problems *and what membership problems are and how they distinguish from other problems, especially with respect to the well-known decision problems. The main topic of this thesis is a complexity analysis of special membership problems, and these membership problems are informally introduced and related to previous work. The definition of the finite recurrent systems, that are considered in Part II, is based on* integer expressions*, that were introduced by Stockmeyer and Meyer. In Chapter 1, the connection is established.*

*The following two chapters are ordinary preliminary chapters as they can be expected in every mathematical text. In Chapter 2, the fundamental notions of graph theory are defined such as directed and undirected graphs themselves, walks, paths, cycles, different subgraph notions and forms of connectivity. Since we will consider walks, paths, cycles in directed as well as in undirected graphs, these and further* *definitions have to be made for both types of graphs. Many more definitions appear in the chapters of Part III, where a lot of graph classes are defined. Here, a special focus is set on acyclic graphs, since these graphs are important throughout the thesis.*

*Chapter 3 is dedicated to complexity theory. Complexity theory and the analysis of problems are based on algorithms. The notion of an algorithm is precised by two different algorithm models: the* random access machine *and the* Turing machine. *Random access machines can be considered mathematical models of real computers. In theory, they are used for analysing algorithms. In contrast, Turing machines developed from considerations about the way computations are executed by humans. They are preferred when fundamental computability questions are treated. Based on both notions complexity notions can be defined. In Chapter 3, we will define the complexity classes that we will use in this thesis, and we will define two reducibilities and complete problems.*

# Chapter 1
# Introduction

The title of the thesis states it precisely: *The complexity of membership problems for finite recurrent systems and minimal triangulations* is studied. Unfortunately, only a few people in the world can imagine what this means! The maybe best known technical term of the title—it is the word “complexity” in my opinion—already produces expectations, and they may lead into the right direction. Complexity theory is the field of theoretical computer science which the contents of this thesis must be located in. (I know well that the algorithms part of the thesis does fit well into the field *algorithms* of theoretical computer science and that the field *algorithms* may be considered different from complexity theory, but one does not have to be so strict.) Since problems are mentioned I can even precise and speak of computational complexity theory. In contrast to structural complexity theory, computational complexity theory is interested in the *complexity* of particular problems instead of whole abstract problem classes. The complexity of a problem is measured by the amount of resources that are consumed by a best algorithm that solves the problem. Usually, there is no ‘best’ algorithm for a problem but a class of algorithms that are good enough in a sense that they can be considered best.

Complexity resource measures can be based on arbitrary computation parameters, but users of real-existing computers mostly ask about time and (memory) space that are needed for carrying out computations. In this thesis, *the complexity of membership problems for finite recurrent systems and minimal triangulations* is analysed in the sense that good algorithms for solving the problems will be presented. Surely, this does not have to mean that they are “best” except for the case that it is obvious. And indeed, this case will happen. If this case does not happen, i.e., if it is not obvious that an algorithm is best possible with respect to current knowledge, it is proved! Everybody knows that such proofs exist only for very very easy problems (and I assure that the problems treated in this thesis are not of the very very easy kind). That is why the proofs for being best algorithm show that better algorithms exist only if something unexpected happens. Such assumptions about unexpected events in complexity theory are popular and common.

## 1.1 Membership problems

So far, the reader knows now that algorithmic solutions of concrete problems have to be expected in this thesis. But still, I did not say what these concrete problems are. Let us start with the classification “membership problem”. Problems in general are simply sets of objects, and in computer science one often restricts to sets of natural numbers or to sets of words over an alphabet, which again can be considered sets of natural numbers. One can imagine many questions that should be solved for a problem, for instance generating an element from the problem set or verifying whether a given object (number) belongs to the set. In the setting of theoretical computer science the first question is rather uninteresting, since for every (non-trivial) problem an algorithm can be constructed that outputs an element of the problem without using almost any resources. The second question, in contrast, is the general description of a decision problem: given an object, is it member of the problem set? A possible solution algorithm is directly associated with the problem and so can have all possible (and finitely encodable) knowledge about the problem. Membership problems are of a similar kind but may be considered generalisations of decision problems: given a set using an appropriate and well-defined representation model and an object, is the object member of the represented set? I hope the difference to decision problems is clear: algorithms solving decision problems implicitly know the set they decide, whereas algorithms solving membership problems learn about the set for which the membership question is to be answered only at input time. To cause a little confusion, membership problems are special decision problems: input set and object for a membership problem are composed into an ordered pair and become input for a decision problem. So, the classification *membership problem* or *decision problem* depends on the point of view and requires fine distinction.

Well, to completely unveil what this thesis is about, it remains to precise the definitions of the sets for which the membership problems will be solved. The membership problems for finite recurrent systems are membership problems about sets of natural numbers, the membership problems for minimal triangulations are membership problems about sets of graphs. A graph is called chordal if it does not contain a chordless cycle of length at least 4. (Some graph notions are used in the following, and I can only refer to later chapters for exact definitions. However, it is not necessary to understand everything here to get an idea of the problem setting.) Not every arbitrary graph is chordal (chordality is not a trivial graph property), but from arbitrary graphs chordal graphs can be obtained by adding edges. This operation, adding edges to a graph to make it chordal, is called *triangulation* or *triangulating a graph* or *making a graph chordal.* The obtained chordal graph is called a triangulation of the start graph. To obtain a *minimal triangulation* of a graph, an inclusion-minimal set of edges is used for triangulating the graph. Such an inclusion-minimal set of edges is called a *minimal fill-in.* So, every graph defines the set of its minimal triangulations, that are the results of minimal fill-in operations. And this is what is required for membership problems. Hence, the membership problem for minimal triangulations is defined as: given a pair $G, H$ of graphs, is $H$ a minimal triangulation of $G$, i.e., is $H$ a member of the set of minimal triangulations of $G$?

The other type of membership problems is based on sets of natural numbers and is of this form: given a natural number and a set of natural numbers represented in a special form, is the given number member of the set? It remains to precise the set representation model. Even though I did not speak about it for the graph problem it is clear that the complexities of the membership problems are mainly determined by the representation model of the set. We will introduce a representation of sets of natural numbers that is based on *integer expressions.* An integer expression is a well-defined expression that is inductively defined as follows:

(a) $\{a\}$ for $a$ a natural number is an integer expression

(b) let $E$ be an integer expression, then $\overline{E}$ is an integer expression

(c) let $E_1$ and $E_2$ be integer expressions, then $(E_1 \cup E_2)$, $(E_1 \cap E_2)$, $(E_1 \oplus E_2)$ and $(E_1 \otimes E_2)$ are integer expressions

(d) further integer expressions do not exist.

Integer expressions were introduced by Stockmeyer and Meyer as the integer (basically natural) analogue of regular expressions [79]. (They did not consider the multiplication operator $\otimes$, which is an addition of a subsequent work by McKenzie and Wagner [61].) The meaning of an integer expression, i.e., the set that is described by an integer expression, is self-explaining except for the definition of the arithmetical operations addition ($\oplus$) and multiplication ($\otimes$). The definition of both operations is extended to sets of numbers in the usual style: for $A$ and $B$ sets of natural numbers, $A \oplus B$ is the set containing the sums of numbers where one operand is taken from $A$ and the other is taken from $B$. Similarly, the product of $A$ and $B$, $A \otimes B$, is defined as the set of products of all pairs of numbers from $A$ and $B$. If set variables are additionally allowed as integer expressions, we obtain parameterised integer expressions, or simply functions. Such functions will be called $\{\cup, \cap, {}^-, \oplus, \otimes\}$-functions, since they are defined using only these operators. (Later, I will not allow singleton sets, i.e., constants, in the definition of $\{\cup, \cap, {}^-, \oplus, \otimes\}$-functions, but this will not make any difference.)

If a $\{\cup, \cap, {}^-, \oplus, \otimes\}$-function $f$ depends on $n$ variables, where fictive variables are allowed, it is called an $n$-ary function. Then, a pair composed of an $n$-ary $\{\cup, \cap, {}^-, \oplus, \otimes\}$-function $f$ and $n$ sets $A_1, \ldots, A_n$ of natural numbers defines a set of natural numbers, which is the result of the application of $f$ to $(A_1, \ldots, A_n)$. Even if $A_1, \ldots, A_n$ are restricted to cardinality only 1 the membership problem becomes difficult, as it was shown by Stockmeyer and Meyer [79]. But I extend the power of expressiveness of integer expressions by defining *finite recurrent systems*. In the setting above, nothing has been said about the structure of the operand sets $A_1, \ldots, A_n$. They shall be the results of $\{\cup, \cap, {}^-, \oplus, \otimes\}$-functions. I precise: a finite recurrent $\{\cup, \cap, {}^-, \oplus, \otimes\}$-system of dimension $n$ is a pair composed of $n$ $n$-ary $\{\cup, \cap, {}^-, \oplus, \otimes\}$-functions and $n$ singleton sets of natural numbers. A set of natural numbers is defined by such a finite recurrent system by iterated application of functions to sets. At first, the functions are simultaneously applied to the given singleton sets, which yields $n$ sets of natural numbers as results. These sets become the inputs of the functions for the second iteration, and again, this yields $n$ sets of natural numbers. Here is an example. Two 2-ary functions are defined as follows:

$$f_1(x, y) =_{\text{def}} (x \cup y)$$
$$f_2(x, y) =_{\text{def}} ((y \oplus y) \cap x) .$$

The two functions are applied to the singleton sets $(\{2\}, \{5\})$ and produce after one

iteration:
$$f_1(\{2\},\{5\}) = (\{2\} \cup \{5\}) = \{2,5\}$$
$$f_2(\{2\},\{5\}) = ((\{5\} \oplus \{5\}) \cap \{2\}) = (\{10\} \cap \{2\}) = \emptyset\,.$$
The results look pretty poor, and even a second iteration does not change the impression that the defined system is a rather uninteresting one:

$$f_1(\{2,5\},\emptyset) = (\{2,5\} \cup \emptyset) = \{2,5\}$$
$$f_2(\{2,5\},\emptyset) = ((\emptyset \oplus \emptyset) \cap \{2,5\}) = \emptyset\,.$$

Since the results of the second iteration are the same as the results of the first iteration, we now know everything about the finite recurrent system $(\langle f_1, f_2\rangle, (2,5))$. A remark: by definition I should write $(\{2\},\{5\})$ instead of $(2,5)$ but I use the more convenient denotation.

In the defined sense, the iteration can be carried out arbitrarily often, and always, this process yields $n$ sets of natural numbers. A function is selected that shall produce an output set, and for a specified number of iterations, the output set produced by the selected function is the set defined by the finite recurrent system after the given number of iterations. For the example, one could say that function $f_2$ shall produce the output. After two iterations, $\emptyset$ is the result. When the iteration process is carried out backwards, i.e., when functions are superposed instead of applying them to input sets, the fixed number iteration defines an integer expression, that represents the output set. Again the example: two iterations yield the integer expression

$$((((\{5\} \oplus \{5\}) \cap \{2\}) \oplus ((\{5\} \oplus \{5\}) \cap \{2\})) \cap (\{2\} \cup \{5\}))\,,$$

which is evaluated to the empty set, which is not obvious. So, one can say that finite recurrent systems for fixed number iterations represent regularly structured integer expressions in a concice form.

I go one step further and define a second model for representing sets of natural numbers by finite recurrent systems. In every iteration the selected output function produces a result. The union of these sets taken over every number of iterations is the set represented by the finite recurrent system. This set may be infinite, and it is not clear whether such sets can be represented by integer expressions.

The complexity of membership problems for finite recurrent systems is studied not only for general finite recurrent $\{\cup, \cap, ^-, \oplus, \otimes\}$-systems but especially for systems that are defined using only subsets of the operator set $\{\cup, \cap, ^-, \oplus, \otimes\}$. The definition of corresponding functions is straightforward. It turns out that the complexities of membership problems for restricted recurrent systems range over a wide field. Already Stockmeyer and Meyer did not consider integer expressions of only the general form but also for a reduced operator set [79]. McKenzie and Wagner extended this work to an almost complete complexity study [61]. The problem setting as well as the results presented here are mainly inspired by these two papers.

In case of the complexity of membership problems for minimal triangulations, there is no direct previous work. The problem, given two graphs $G$ and $H$, is $H$

a minimal triangulation of $G$, has been considered before only as an application. There exists an easy characterisation of minimal triangulations of a graph: graph $H$ is a minimal triangulation of a graph $G$ if and only if $H$ is chordal, is a spanning supergraph of $G$ and the deletion of every edge of $H$ that is not an edge of $G$ yields a graph that is not chordal [76]. Ibarra considered the question: how fast can it be decided whether a chordal graph remains chordal after deletion of a specified edge [44]? He showed that, given the chordal graph using an appropriate representation, this question can be decided in time $\mathcal{O}(n)$, where $n$ denotes the number of vertices of the graph. Given a graph $G$ and a graph $H$, where $H$ is a chordal spanning supergraph of $G$ that contains $f$ additional edges, Ibarra's results show that it can be decided in time $\mathcal{O}(fn + m)$ whether $H$ is a minimal triangulation of $G$, where $m$ denotes the number of edges of $G$. The algorithm is easy: for every additional edge of $H$, ask whether the deletion of this edge results in a chordal graph. If all questions are answered 'no', $H$ is a minimal triangulation of $G$ due to the cited characterisation. It is only remarked that time $\mathcal{O}(n + m + f)$ is needed to generate the required representation for $H$.

There are further algorithms that also answer the question about being minimal triangulation. But their answers are indirect ones. The original problem is as follows: given a graph $G$ and a graph $H$, where $H$ is a chordal spanning supergraph of $G$, compute a minimal triangulation of $G$ that is a subgraph of $H$. This problem was dealt with by Blair, Heggernes, Telle [6] and Dahlhaus [23]. Their algorithms work in time not better than $\mathcal{O}(nm)$ where $n$ and $m$ denote the numbers of vertices and edges, respectively, of graph $G$. So, if the output graph is equal to input graph $H$, $H$ is already a minimal triangulation of $G$.

Similar to the case of membership problems for finite recurrent systems, I will not consider the membership problem for minimal triangulations for arbitrary graphs but for restricted graph classes. These graph classes are the class of $2K_2$-free graphs, the class of permutation graphs and the class of AT-free claw-free graphs. Particularly the class of permutation graphs is a well-studied graph class, and many problems are easy for these graphs (see, for instance and for standard problems, Golumbic's book [35]). To name the major results: for all three graph classes, I can present linear-time algorithms that solve the membership problems, if it is not necessary to verify that input graph $G$ belongs to the requested graph class but it is simply assumed. The algorithms are based on characterisations of minimal triangulations of graphs from these graph classes that allow efficient representations of minimal triangulations, and considerable amount of work is devoted to obtain such characterisations. The above-cited characterisation of minimal triangulations of arbitrary graphs does not suffice.

## 1.2 Composition rules and outline of the thesis

The present thesis is partitioned into three major Parts and eleven Chapters, among which the first one is currently read. Every chapter is partitioned into several sections. Part I is mainly dedicated to the introduction and the presentation of fundamental definitions and results from the fields of graph theory (Chapter 2) and complexity theory (Chapter 3). If the reader feels familiar with basic notions of both fields Chapters 2 and 3 can be skipped. I tried to recall definitions that seem standard but may nevertheless differ in details from definitions by other authors when they are needed in Parts II and III, so that it shall be easier to remember and understand. The following shall summarise the main parts of the thesis and highlight important results and achievements.

***Part II*** is dedicated to the complexity of membership problems for finite recurrent systems. This part consists of four chapters.

*Chapter 4*
Membership problems for finite recurrent $\{\oplus\}$- and $\{\otimes\}$-systems turn out to be related to problems that ask for the numbers of walks in directed graphs. So-called numbers-of-walks problems are introduced, and their complexities are determined. All problems can be solved in polynomial time, which makes them easy in a certain sense. Some of them are even complete for nondeterministic logarithmic space, and this result is needed later for proving hardness of membership problems. Another problem is introduced, that asks for satisfiability of a system of congruence equations. This problem can be considered a decisional generalisation of the Chinese Remainder Theorem and turns out complete for nondeterministic polynomial time.

*Chapter 5*
Finite recurrent systems are formally introduced, and membership problems for finite recurrent $\{\cup, \cap, ^{-}, \oplus, \otimes\}$-systems are defined. Upper complexity bounds for the general membership problems are proved. I have only spoken of solving membership problems, which implies the existence of algorithms in the usual context. The upper bound then is disappointing in the sense that it does not provide a "real"—executable—algorithm but only an algorithm that has oracle access to the halting, or a related, problem.

*Chapter 6*
Many membership problems for restricted finite recurrent systems are classified with respect to their complexities. In most cases, completeness results are obtained for the complexity classes NL, NP and PSPACE. Among the results of this chapter, containment in NP of membership problems for finite recurrent $\{\cap, \oplus\}$-systems requires a number of partial results. Even more complex is the proof of containment in PSPACE of membership problems for finite recurrent $\{\cup, \oplus, \otimes\}$-systems.

*Chapter 7*
Three membership problems are considered for which there do not exist decision algorithms. This particularly shows that the upper complexity bound obtained in

Chapter 5 is not so bad as it seemed. The result is obtained by reducing a known undecidable problem to the membership problems. The chosen undecidable problem is the satisfiability problem for Diophantine equations.

***Part III*** is dedicated to the complexity of membership problems for minimal triangulations. This part consists of four chapters.

*Chapter 8*
The class of chordal graphs is got to know. Alternative characterisations of chordal graphs are presented. Minimal triangulations are defined and two characterisations, that are used in later chapters, are stated. The membership problem for minimal triangulations of arbitrary graphs is defined.

*Chapter 9*
The class of $2K_2$-free graphs is defined, and some properties are shown. Two characterisations of minimal triangulations of $2K_2$-free graphs are proved. One result characterises the set of minimal triangulations of $2K_2$-free graphs in general, and the other result is a characterisation of the minimal triangulations of a single $2K_2$-free graph. Using the second characterisation, the membership problem for minimal triangulations of $2K_2$-free graphs is solved.

*Chapter 10*
The class of AT-free graphs and further subclasses are defined. In particular, the class of permutation graphs is a subclass of the class of AT-free graphs. The central result of the chapter is a characterisation of the minimal triangulations of a single permutation graph. This characterisation is based on potential maximal cliques, that are characterised by an easy property. Besides the solution of the membership problem for minimal triangulations of permutation graphs, linear-time algorithms for computing treewidth and minimum fill-in of permutation graphs are important results.

*Chapter 11*
The central result, a characterisation of minimal triangulations of AT-free claw-free graphs, is based on a breadth first search style algorithm, that is called `min-LexBFS`. This algorithm works on graphs and generates vertex orderings for input graphs. These orderings are studied: they are characterised and properties are proved. An important auxiliary result is that `min-LexBFS` finds a moplex of the input graph, that is a special clique whose neighbourhood is a minimal separator. Another auxiliary result is a recognition algorithm for a chordal subclass of AT-free claw-free graphs, that is based on `min-LexBFS`.

The thesis is concluded by a list of cited papers (*Bibliography*), a list of used mathematical symbols and definitions (*Mathematical symbolism*), a list of defined decision problems and an index.

I tried to create two parts, I speak of Thesis Parts II and III, that are comparable. Many further results could have been proved. But due to the size the present thesis already has, I decided to restrict on results that are necessary for achieving the theorems. Only in case of Part III, results beyond the solution of membership problems are considered, since they are of major importance in the context of minimal triangulations. On the contrary, I did not pay too much attention to space. In some cases, it was necessary to reprove theorems known from the literature, since they provided useful inside or, particularly in Part III, an algorithm to which it is referred. So, I had the idea to reprove further cited theorems, if the proofs are not too space-consuming. I like it this way, and I strongly believe that the thesis profits much from this decision.

Finally, the dependencies among the different parts and chapters of this thesis, as I see them, are shown in Figure 1. Since Chapter 4 in Part II contains only auxiliary results with respect to the main aspects of the thesis, it is not considered central. The filled regions mark the parts of the thesis containing the main results or definitions of new objects, as it is the case for Chapter 5. Chapter 8 is the minimal triangulations analogue of Chapter 5. But since Chapter 8 is dedicated to the presentation of only preliminary definitions and results, it is also not considered central.

## 1.3 Publishment

Some of the results that are contained in this thesis have already been presented at conferences or been published in a refereed journal. The main results of Chapter 11 were published in

- D. Meister, *Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs*, Discrete Applied Mathematics 146, pp. 193–218, 2005.

Many results of Chapter 10 as well as the main parts of the Chapters 5 and 6 were presented at international conferences and published in conference proceedings:

- D. Meister, *Computing Treewidth and Minimum Fill-in for Permutation Graphs in Linear Time*, Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 3787, pp. 91–102, Springer-Verlag, 2005
- D. Meister, *Decidable Membership Problems for Finite Recurrent Systems over Sets of Naturals*, Proceedings of the 15th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Computer Science 3623, pp. 80–91, Springer-Verlag, 2005.

The managing boards of the Department of Computer Science of the Bayerische Julius-Maximilians-Universität Würzburg edit a technical reports series named "Forschungsberichte". Most of the results of the thesis, the already published results included, are contained in technical reports; nos. 302, 328, 336, 361, 369.

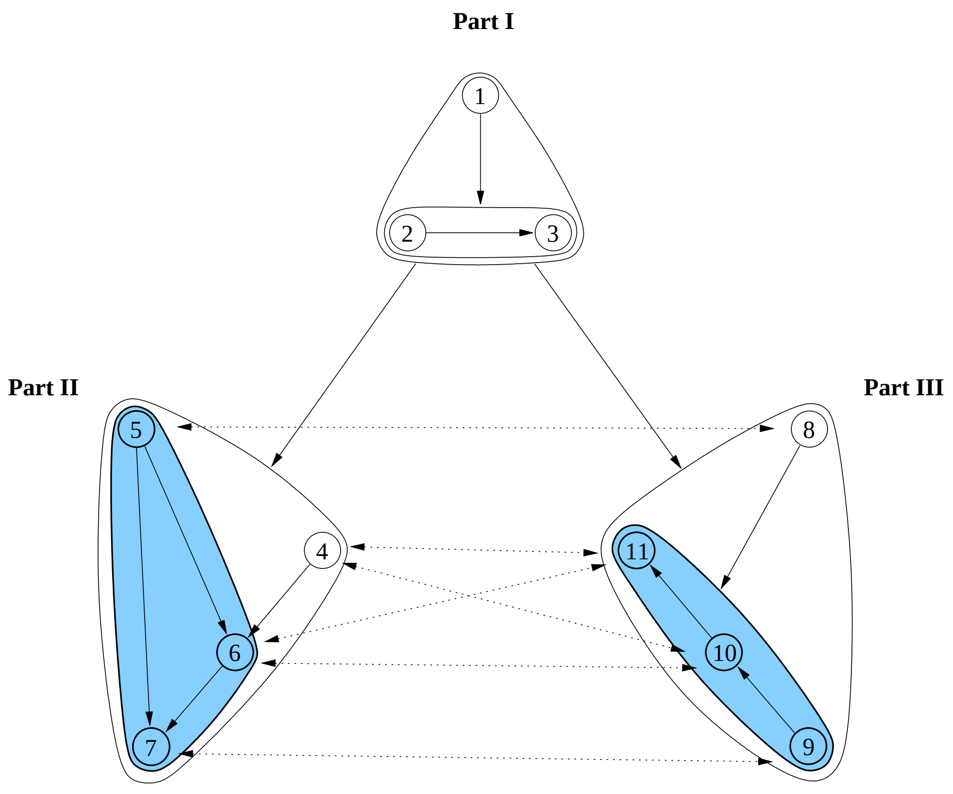


**Figure 1** How are parts and chapters of the thesis related to each other? The filled regions mark the chapters of the thesis containing the main results. The dotted arrows show conceptional correspondences between the chapters of Part II and Part III.

# Chapter 2
# Graph-theoretic fundamentals

A simple undirected graph $G$ is an ordered pair $(V, E)$ of sets where $V$ is the set of vertices of $G$ and $E$ is the set of edges of $G$. An edge of $G$ is a set containing exactly two elements from $V$, i.e., two vertices of $G$. An edge $\{u, v\}$, where $u$ and $v$ are distinct vertices, is briefly denoted as $uv$. If $uv$ is an edge of $G$, we say that $u$ and $v$ are *adjacent* or that $u$ is *adjacent with* $v$. For an edge $uv$ of $G$, we call $u$ and $v$ the *endpoints* of $uv$. The vertex and edge sets of $G$ are also denoted as $V(G)$ and $E(G)$, respectively. If graphs are intended to be not simple, their edge sets are not sets of subsets of vertex sets but families of subsets of vertex sets. Here, we only consider simple graphs. If $V$ is a finite set, $G$ is a finite graph. Here, we only consider finite graphs. We only remark that a graph which is not simple may have a finite vertex set but infinitely many edges. For a given graph $G$, $n$ and $m$ usually denote the numbers of vertices and edges of $G$, respectively. It will not be necessary to add indices to $n$ or $m$.

A simple directed graph $G$ is an ordered pair $(V, A)$ of sets where $V$ is the set of vertices of $G$ and $A$ is the set of arcs of $G$. An arc of $G$ is an ordered pair of vertices of $G$, and it is not required that both vertices are distinct from each other. Arcs with both vertices the same are called *loops*. If $(u, v)$ is an arc of $G$, we say that $u$ is *adjacent to* $v$. In this case, $u$ is a *predecessor* of $v$, and $v$ is a *successor* of $u$. Furthermore, $(u, v)$ is an *in-coming* arc of $v$ and an *out-going* arc of $u$. Vertex and arc sets of $G$ are also denoted as $V(G)$ and $A(G)$, respectively. We will not consider directed graphs that are not simple. When we speak of "graphs", we always mean undirected graphs. Furthermore, we define that a graph on $n$ vertices (this will always mean that the graph has exactly $n$ vertices) has vertex set $\{1, \ldots, n\}$. Then, the numbers are the *names* of the vertices, which clearly must be considered synonymous for being vertex itself. Note that this does not mean that every graph has vertex set $\{1, \ldots, n\}$ for some number $n$, but we can assume that it is always a (finite) subset of $\mathbb{N}$. (Usually, vertex sets can be sets of arbitrary objects.) In special cases, we attach marks to vertices, edges and arcs, and these marks are called "labels" then.

Graphs appear in many contexts. Especially directed graphs represent binary relations over a finite set. Undirected graphs can be considered representations of binary symmetric relations over a finite set, that may be reflexive or irreflexive (we do not allow loops in undirected graphs). This connection to binary relations is a good explanation why graphs are used in so many different areas of computer science. And this is also why purely abstract problems on graphs are studied and solved. Most of them are motivated by real-world problems. To name only a few, colouring the vertices of a graph with the least possible number of different colours, determining largest sets of mutually adjacent or mutually non-adjacent vertices are among the most fundamental problems on graphs.

An important question in connection with computations touches representation aspects. Let $G$ be an undirected graph. For each vertex $x$ of $G$, the *adjacency list* attached to $x$ is the set of vertices of $G$ that are adjacent with $x$. 'List' usually means that this set is implemented using a list structure. Hence, if a graph is given by "adjacency lists", this means that the graph is represented by a list of adjacency lists. In case of directed graphs, *adjacency list representation* is defined analogously. There are further graph representations—some of them are suitable for general graphs, others are suitable only for special graphs. If a representation model for graphs is not specified explicitly, we always assume adjacency list representation.

This chapter contains all information—mainly definitions—about graphs that we will need in this thesis. We will define walks, paths, cycles in directed as well as undirected graphs. From this, we will derive different connectivity notions, and we will define graph parameters. A special subclass of directed graphs is discussed at the end of the chapter.

## 2.1 Subgraphs, isomorphism and neighbourhoods

An outstanding property of (finite) graphs is the possibility of visualization. In other words, and more precisely, graphs can be drawn on a sheet of paper, and many notions can be described on such drawings. So, the notion of a graph is easily explicable by examples. Figure 2 shows many graphs with only a few vertices. To understand the drawings, vertices are represented by dots and adjacency is expressed by line segments between two dots, i.e., vertices. Vertices (dots) that are not joined by a direct line segment are meant non-adjacent. The two special graphs are sample graphs that will be needed later. Some graphs also have names such as the claw. Graph $C_4$ is sometimes called *square*. Among the depicted graphs, two graphs can be considered maximal: $4K_1$ and $K_4$. The former graph does not contain any edge, and the latter one contains all possible edges. To give names, the *blank* graph on $n$ vertices is the graph with vertex set $\{1, \ldots, n\}$ and empty edge set, and the *complete* graph on $n$ vertices is the graph with vertex set $\{1, \ldots, n\}$ and all possible edges. So, $4K_1$ and $K_4$ are the blank and complete graphs on four vertices, respectively. The *empty graph* is the graph without any vertex and edge.

Let $G = (V, E)$ and $H = (W, F)$ be graphs. We say that $G$ is a *subgraph* of $H$, denoted as $G \subseteq H$, if $V \subseteq W$ and $E \subseteq F$. If one of the inclusions is strict, $G$ is a *proper* subgraph of $H$, denoted as $G \subset H$. If $G \subseteq H$ we call $H$ a *supergraph* of $G$. If $G \subseteq H$ and $V = W$, we call $G$ a *spanning* subgraph of $H$ and $H$ then is a *spanning* supergraph of $G$. If $G$ is a subgraph of $H$ and contains all possible edges of $H$, $G$ is an *induced* subgraph of $H$. Precisely, $G$ is the subgraph of $H$ *induced* by $V$, denoted as $H[V]$, if, for every pair $u, v$ of vertices of $G$, $uv \in E$ if and only if $uv \in F$. Induced subgraphs are the results of vertex deletion operations. We say that $G$ and $H$ are equal if $V = W$ and $E = F$. This is equal to $G \subseteq H$ and $H \subseteq G$. A relaxation of equality is provided by isomorphism. We say that $G$ and $H$ are *isomorphic*, denoted

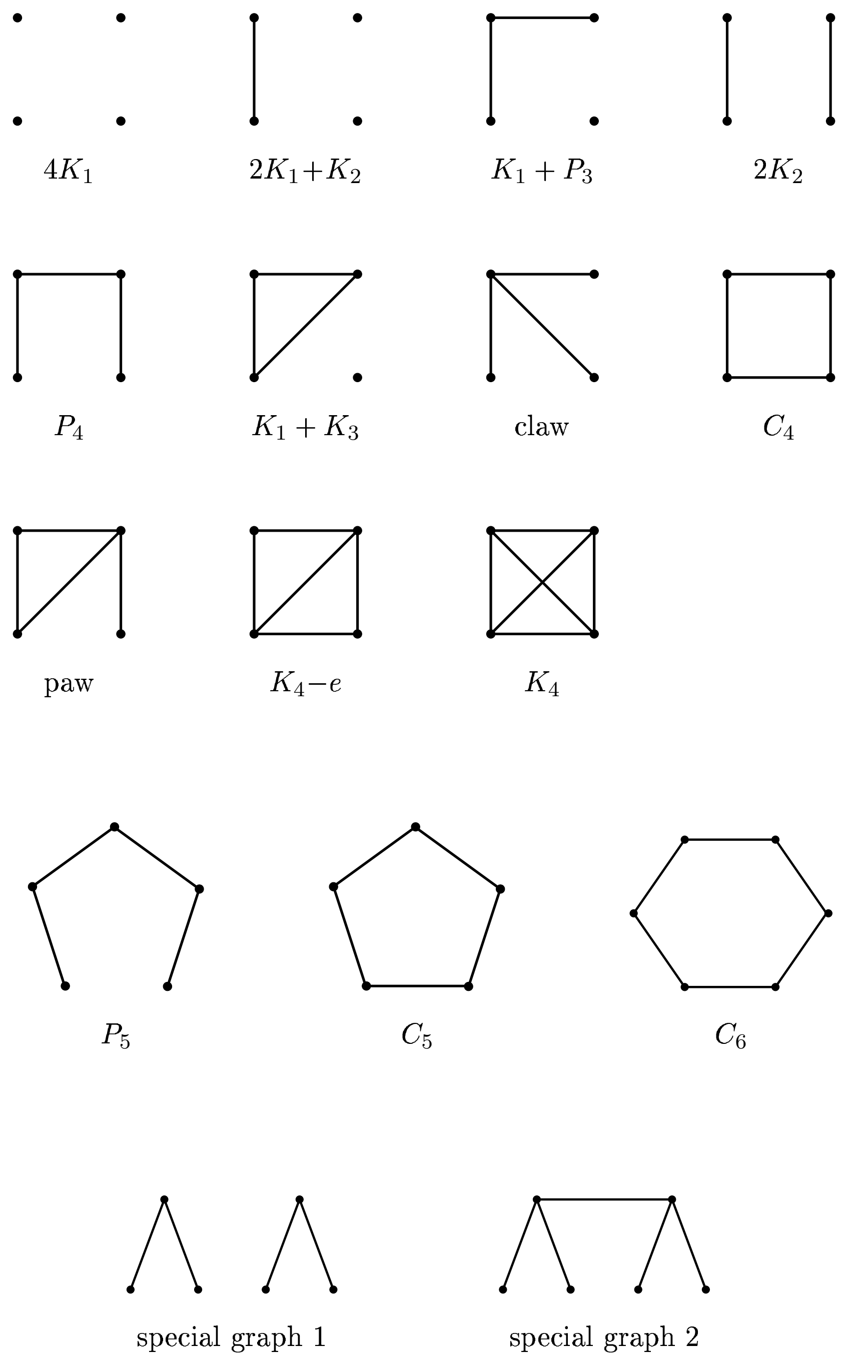


**Figure 2** Many small graphs.

as $G \cong H$, if and only if $|V| = |W|$ and there is a bijective function $\varphi$ from $V$ to $W$ such that, for every pair $u, v$ of vertices of $G$, $uv \in E \Leftrightarrow \varphi(u)\varphi(v) \in F$. Informally, $G$ and $H$ are isomorphic if $H$ can be obtained from $G$ by simply renaming vertices.

Let $G = (V, E)$ be a graph, and let $u$ be a vertex of $G$. The *neighbourhood* of $u$ in $G$, denoted as $N_G(u)$, is the set of vertices of $G$ that are adjacent with $u$. The *closed neighbourhood* of $u$, denoted as $N_G[u]$, is the set $N_G(u) \cup \{u\}$. To emphasise the distinction, the neighbourhood of a vertex is also called its *open* neighbourhood. Let $A \subseteq V$ be a set of vertices of $G$. The neighbourhood of $A$ is defined as $N_G(A) =_{\text{def}} \bigcup_{u \in A} N_G(u) \setminus A$, and the closed neighbourhood of $A$ is $N_G[A] =_{\text{def}} N_G(A) \cup A$. If $u$ is adjacent with every other vertex of $G$, i.e., if $N_G[u] = V$, we call $u$ a *universal* vertex. Neighbourhood can also be defined for edges. Let $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ be edges of $G$. We say that $e_1$ and $e_2$ are *adjacent* if $|\{u_1, v_1\} \cap \{u_2, v_2\}| = 1$, i.e., if $e_1$ and $e_2$ are different edges with a common endpoint. If vertex $u$ is an endpoint of edge $e$, we say that $u$ and $e$ are *incident*. For a set $A$ of vertices of $G$, we call $A$ a *clique* in $G$ if every pair of vertices in $A$ is adjacent in $G$. In other words, $A$ induces a complete subgraph of $G$. If every pair of vertices in $A$ is non-adjacent in $G$, $A$ is an *independent set* in $G$. In this case, $A$ induces a blank subgraph of $G$. A clique of $G$ is a set of vertices of $G$ that is a clique in $A$. A clique of $G$ is *maximal* if it is not properly contained (by inclusion) in any other clique of $G$. The maximum size of a clique of $G$ is called the *clique number* of $G$ and denoted as $\omega(G)$. Similarly, maximal independent sets of $G$ are defined as well as the *independent set number* of $G$ denoted as $\alpha(G)$. The *complement* of $G$, denoted as $\overline{G}$, is the graph on the vertex set of $G$, and two vertices $u$ and $v$ of $\overline{G}$ are adjacent in $\overline{G}$ if and only if they are non-adjacent in $G$. It holds that $\alpha(G) = \omega(\overline{G})$, since independent sets of $G$ are cliques of $\overline{G}$, and vice versa.

In many cases, it is convenient to define sub- or supergraphs of a given graph by application of basic operations. Let $G = (V, E)$ be a graph. Let $S \subseteq V$ be a set of vertices of $G$. By $G \setminus S$, we denote the subgraph of $G$ that is obtained from $G$ by deleting the vertices in $S$ and all edges incident with vertices in $S$. In other words, $G \setminus S = G[V \setminus S]$ is the subgraph of $G$ induced by $V \setminus S$. Let $u$ be a vertex of $G$. By $G{-}u$, we denote the subgraph of $G$ that is obtained from $G$ by deleting vertex $u$ and all edges incident with $u$, i.e., $G{-}u = G \setminus \{u\} = G[V \setminus \{u\}]$. Let $e$ be an edge of $G$. By $G{-}e$, we denote the subgraph of $G$ obtained by deletion of $e$. For $v$ a vertex that is not a vertex of $G$ and $f$ an edge that is not an edge of $G$, $G{+}v$ and $G{+}f$ denote the graphs obtained from $G$ by adding vertex $v$ and edge $f$, respectively. Note that in case of $G{+}v$ the neighbourhood of $v$ needs further precision. In some cases, it may be already defined by context. Let $F$ be a set of edges suitable for $G$ such that $F \cap E(G) = \emptyset$. Then, $G \cup F =_{\text{def}} (V(G), E(G) \cup F)$. Let $H$ be another graph. The *disjoint union* of $G$ and $H$, denoted as $G \cup H$, is defined as follows: the vertex set of $G \cup H$ has $|V(G)| + |V(H)|$ many vertices, and they are partitioned into two sets one of which corresponds to $V(G)$, the other to $V(H)$. Two vertices of $G \cup H$ are adjacent if and only if they are in the same partition class and the corresponding vertices in $G$ or $H$ are adjacent. Let $u$ and $v$ be vertices of $G$. Graph $G'$ is the result of *glueing together* $u$ and $v$, if all vertices adjacent with $v$ in $G$ become adjacent also with $u$ and $v$ is deleted. Note that, if $u$ and $v$ are adjacent, this same edge is ignored.

## 2.2 Walks, paths, cycles and connectivity

For the definitions in this section, we often have to distinguish the cases directed or undirected graph. We begin with definitions for undirected graphs. Let $G = (V, E)$ be an undirected graph. Let $k \geq 0$, and let $u$ and $v$ be vertices of $G$. Let $x_0, \dots, x_k$ be $k + 1$ vertices of $G$, that do not have to be distinct. The sequence $W = (x_0, \dots, x_k)$ is a *$u, v$-walk* in $G$ if $x_0 = u$ and $x_k = v$ and $x_i x_{i+1} \in E$ for every $i \in \{0, \dots, k - 1\}$. A $u, v$-walk $W$ in $G$ is a *$u, v$-path* in $G$ if $W$ does not contain multiple occurrences of vertices; in particular, $k < n$. Finally, a $u, v$-path in $G$ is a *cycle* in $G$ if $uv \in E$. The *length* of a walk or a path is one less the number of contained vertices. The *length* of a cycle is the number of contained vertices. Alternatively, one can say that the length of a walk, a path, a cycle is the number of appearing edges (where "chords" are not counted). Let $P = (x_0, \dots, x_k)$ be a path in $G$. The edge $xz$ of $G$ is a *chord* in $P$ if there are $i, j \in \{0, \dots, k\}$, $i \neq j$, such that $x_i = x$ and $x_j = z$ and $|i - j| \geq 2$. In other words, chords are edges joining non-consecutive vertices of $P$. As a special case, if $x_0 x_k$ is an egde of $G$ (and $k \geq 2$), then $x_0 x_k$ is a chord in $P$. Chords in cycles are defined similarly. Let $C = (x_1, \dots, x_k)$ be a cycle in $G$. The edge $xz$ of $G$ is a *chord* in $C$ if there are $i, j \in \{1, \dots, k\}$, $i \neq j$, such that $x_i = x$ and $x_j = z$ and $2 \leq |i - j| \leq k - 2$. We have to be careful, since $x_1 x_k$ is certainly not a chord in $C$. A path or a cycle in $G$ is *chordless* if $G$ does not contain an edge that is a chord in the path or the cycle. Let $k \geq 1$. The *chordless path* on $k$ vertices is the graph $(\{1, \dots, k\}, \{12, 23, \dots, (k - 1)k\})$. When we speak of "chordless paths" as induced subgraphs then we mean a graph isomorphic to a chordless path. Similar definitions hold for chordless cycles.

Now, let $G = (V, A)$ be a directed graph. Walks, paths and cycles in $G$ are defined analogously to the undirected case. Let $k \geq 0$, and let $u$ and $v$ be vertices of $G$. Let $x_0, \dots, x_k$ be $k + 1$ vertices of $G$. The sequence $W = (x_0, \dots, x_k)$ is a *$u, v$-walk* in $G$ if $x_0 = u$ and $x_k = v$ and $(x_i, x_{i+1}) \in A$ for every $i \in \{0, \dots, k - 1\}$. If $W$ does not contain a vertex twice, it is a *$u, v$-path* in $G$. And if $(x_k, x_0) \in A$, $W$ is a *cycle* in $G$. Lengths of walks, paths, cycles are defined similar to the lengths in undirected graphs.

Let $G = (V, E)$ be an undirected graph. We say that $G$ is *connected* if, for every pair $u, v$ of vertices of $G$, there is a $u, v$-path in $G$. Note that this condition is equivalent to requiring walks. If $G$ is not connected, maximal connected (induced) subgraphs of $G$ are called *connected components* of $G$. Hence, connected components of graphs are graphs. The *distance* of vertices $u$ and $v$ of $G$, denoted as $\text{dist}_G(u, v)$, is the length of a shortest $u, v$-path in $G$; if $u$ and $v$ are in different connected components, i.e., if there is no $u, v$-path in $G$, we set $\text{dist}_G(u, v) = \infty$. Note that in undirected graphs, $\text{dist}_G(u, v) = \text{dist}_G(v, u)$. Finally, the *diameter* of $G$, denoted as diam$(G)$, is the maximal distance between any pair of vertices of $G$; in particular, if $G$ is not connected, diam$(G) = \infty$.

In case of directed graphs, we distinguish two notions of connectivity. Let $G = (V, A)$ be a directed graph. We say that $G$ is *strongly connected* if, for every pair $u, v$ of vertices of $G$, there are a $u, v$-walk as well as a $v, u$-walk in $G$. In contrast, if, for every pair $u, v$ of vertices of $G$, there is a $u, v$-walk or a $v, u$-walk, we say that $G$ is *weakly connected.* Note that strong connectivity implies weak connectivity, but not vice versa in general.

## 2.3 Topological orderings and acyclic graphs

Let $G = (V, E)$ be a graph that may be directed or undirected. A *vertex ordering* for $G$ is an ordering of the vertices of $G$. Let $G$ have $n$ vertices, and let $x_1, \ldots, x_n$ be the vertices of $G$. Furthermore, let $\pi$ be a bijective mapping from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$. We will call mappings like $\pi$ also *permutations* later. Then, $\sigma = \langle x_{\pi(1)}, \ldots, x_{\pi(n)} \rangle$ defines a vertex ordering for $G$. For any pair $u, v$ of vertices of $G$, $u \prec_\sigma v$ if and only if $i < j$ where $u = x_{\pi(i)}$ and $v = x_{\pi(j)}$. We also say that $u$ is *to the left* of $v$ with respect to $\sigma$. We also use the term *leftmost vertex with respect to $\sigma$*. A vertex $x$ is leftmost with respect to $\sigma$ and a given property, if $x$ has this property and there is no vertex $z$ such that $z$ has the property and $z \prec_\sigma x$. For $i \in \{1, \ldots, n\}$, $\sigma(i) = x_{\pi(i)}$ is the vertex at position $i$ in $\sigma$ and $\sigma^{-1}(x_{\pi(i)}) = i$ is the position in $\sigma$ of vertex $x_{\pi(i)}$. The *reversed ordering* of $\sigma$ is $\vec{\sigma} = \langle x_{\pi(n)}, \ldots, x_{\pi(1)} \rangle$. Vertex orderings will play a significant role in Part III of this thesis, where special orderings provide interesting characterisations of graph classes. A well-known ordering characterisation of acyclic graphs is given in the following.

Among the directed graphs, the class of acyclic graphs plays a prominent role. Different to any other class of directed graphs, walks in acyclic graphs are always paths. In a certain sense (that may be clearer later), acyclic graphs can be considered uni-directed.

**Definition 1** *Let $G = (V, A)$ be a directed graph. We say that $G$ is **acyclic** if and only if $G$ does not contain a cylce.*

In other words, in an acyclic graph it is not possible to find a walk that contains a vertex twice. As a rule, when we speak of acyclic graphs, we assume the graph to be directed. In case of undirected graphs, the corresponding notion of acyclicness defines the class of forests (disjoint unions of trees). We look for an easier characterisation of acyclic graphs. In fact, this characterisation will be by special vertex orderings.

**Definition 2** *Let $G = (V, A)$ be a directed graph, and let $u$ be a vertex of $G$. We say that $u$ is a **source** of $G$ if and only if $u$ has no in-coming arcs. Analogously, $u$ is called a **sink** of $G$ if and only if $u$ has no out-going arcs.*

**Lemma 2.1 (folklore)**
*Let $G = (V, A)$ be an acyclic graph. Then, $G$ contains a source.*

**Proof:** We give an algorithm that surely finds a source of $G$. Let $x$ be any vertex of $G$. If $x$ is a source, the proof is done. Otherwise, $G$ has a vertex $x'$ that is adjacent to $x$. Either $x'$ is a source of $G$, or the algorithm procedes. Since $G$ is acyclic, $G$ contains no cycle, hence the described algorithm must end with a vertex without in-coming arc, i.e., with a source.

■

An immediate question is whether the statement of Lemma 2.1 can be strengthened to a characterisation of acyclic graphs. This is certainly not possible. But Lemma 2.1 implies such a characterisation.

**Definition 3** *Let $G = (V, A)$ be a directed graph, and let $\sigma$ be a vertex ordering for $G$. We say that $\sigma$ is a* ***topological ordering*** *for $G$ if and only if, for every pair $u, v$ of vertices of $G$, holds:*

$$(u, v) \in A \quad \Longrightarrow \quad u \prec_\sigma v\,.$$

**Theorem 2.2 (folklore)**
*A directed graph is acyclic if and only if it has a topological ordering.*

**Proof:** Let $G = (V, A)$ be a directed graph, and let $\sigma$ be a topological ordering for $G$. Suppose $G$ is not acyclic, i.e., $G$ contains a cycle $C = (x_1, \ldots, x_r)$, $r \geq 1$. Without loss of generality, $x_1$ is the vertex of $C$ leftmost with respect to $\sigma$. In other words, $x_1 \prec_\sigma x_i$ for every $i \in \{2, \ldots, r\}$. If $r = 1$ then $C = (x_1)$, and $(x_1, x_1)$ is an arc in $G$. Since $x_1 \not\prec_\sigma x_1$ and $\sigma$ is a topological ordering for $G$, $r \geq 2$. It holds that $(x_r, x_1) \in A$. Vertex $x_r$ is the predecessor of $x_1$ on $C$. According to the definition of topological orderings, $x_r \prec_\sigma x_1$, which contradicts the choice of $x_1$. Thus, $G$ cannot contain a cycle, and $G$ is acyclic.

For the converse, let $G$ be acyclic. We prove the existence of a topological ordering for $G$ by induction over the number of vertices of $G$. If $G$ has exactly one vertex, say $x$, $\langle x \rangle$ is a topological ordering for $G$. So, let $G$ have $n \geq 2$ vertices. By Lemma 2.1, $G$ contains a source, say vertex $x$. Consider graph $G' =_{\text{def}} G - x$. It is clear that $G'$ is also acyclic, and applying the induction hypothesis, $G'$ has a topological ordering, say $\sigma' = \langle u_1, \ldots, u_{n-1} \rangle$. Note that all neighbours of $x$ in $G$ are contained in $\{u_1, \ldots, u_{n-1}\}$. Then, $\sigma =_{\text{def}} \langle x, u_1, \ldots, u_{n-1} \rangle$ is a topological ordering for $G$.

■

# Chapter 3
# Complexity-theoretic fundamentals

A letter is a symbol. An alphabet is a finite and non-empty set of letters. Let $\Sigma$ be an alphabet. A word over $\Sigma$ is a finite sequence of letters from $\Sigma$. The set of words over $\Sigma$ is denoted as $\Sigma^*$. The length of a word is the number of symbols it contains. A language over $\Sigma$ is a set of words over $\Sigma$. Problems over $\Sigma$ are languages over $\Sigma$. Problems are the main objects theoretical computer science and most particularly complexity theory (in the broadest sense) deal with. Two fundamental questions about problems are in the focus of complexity theory: what is a (very) good algorithm for a given problem and how difficult is a given problem with respect to other problems?

Structural complexity theory developed from basic considerations about concrete problems. In structural complexity theory, concrete problems are of only minor interest; rather properties of classes of problems are investigated and problem classes are related to each other. Many properties that define complexity classes are based on a fixed notion of algorithms: the Turing machine. There are also other formal definitions of algorithms—random access machines are of fundamental importance in the field of "real" algorithms, i.e., algorithms that are designed for being implemented on real-world computing devices. Nevertheless, complexity class definitions are sufficiently robust in most cases such that they do not really rely on the underlying algorithm definition.

In this chapter, we will define random access machines and Turing machines, which our theoretic considerations are based on. Especially for the results of Part II of this thesis, special complexity classes are needed, and they are defined here. Finally, we present the notions that are needed for relating and comparing problems, namely we will define reducibilities and hardness and completeness notions.

## 3.1 Random access and Turing machines

There are many approaches to formalize algorithms. In theoretical computer science, two models are of fundamental importance when concrete problems are considered. In the field of algorithms the random access machine is used, in the field of (structural and computational) complexity theory the Turing machine is used. A random access machine is considered a good mathematical modelization of real-existing computers based on von-Neumann architecture. Turing machines are more or less purely mathematical objects, and they model computations that work on the symbolic (representation instead of information) level. Most text books about theoretical computer science introduce Turing machines and random access machines; we refer to Papadimitriou's book about computational complexity for further information [68].

A *random access machine* consists of an infinite set of registers and a programme.

The registers contain natural numbers of arbitrary size, and they are denoted as BR and R$i$ where $i$ is a natural number. Register BR is a special register and indicates the next programme instruction that is to be executed. Programmes of random access machines are finite sequences of instructions. We distinguish arithmetical instructions, transportation instructions and control instructions. Control instructions are jump instructions and the final instruction, and they affect programme execution.

`goto` $k$
: *programme execution continues with instruction number k*

`if` R$i = 0$ `then goto` $k$
: *if register* R$i$ *contains 0, the programme execution continues with instruction number k; otherwise programme execution continues with the following instruction*

`stop`
: *programme execution stops*

The first instruction is an unconditional jump, whereas the second instruction is a conditional jump. It is clear that the conditional jump may be restricted to depend only on a fixed register.

Transportation instructions move data between registers. We distinguish direct and indirect addressing.

`mov` $i, j$
: *move the content of register* R$j$ *to register* R$i$*; continue with the following instruction*

`mov` $i, [j]$
: *move the content of the register whose number is contained in register* R$j$ *to register* R$i$*; continue with the following instruction*

`mov` $[i], j$
: *move the content of register* R$j$ *to the register whose number is contained in register* R$i$*; continue with the following instruction*

Finally, arithmetical instructions modify contents of registers. We restrict to the basic operations addition and subtraction, and a register can be initialized with a constant.

`store` $i, k$
: *store number k in register* R$i$*; continue with the following instruction*

`add` $i, j$
: *add the content of register* R$j$ *to the number in register* R$i$*; the content of register* R$i$ *after execution is the result of the addition; continue with the following instruction*

`sub` $i, j$
: *subtract the content of register* R$j$ *from the content of register* R$i$*; if the content of* R$j$ *is greater than the content of* R$i$*, the result will be 0; the*

*content of* R$i$ *after execution is the result of the subtraction; continue with the following instruction*

Basically, random access machines compute functions over the natural numbers. Input parameters are stored in the first registers in appropriate order, and programme execution starts with the first instruction of the programme sequence. For simplicity, the first instruction has number 0. Execution continues as long as possible. The content of register BR is the number of the next instruction in the sequence that is to be executed; it is modified by the execution of every single instruction. If the `stop` instruction is encountered or BR contains a number that is larger than the largest instruction number, programme execution stops. The result of the execution is contained in register R0.

A different computation model is defined by Turing machines. *Turing machines* consist of a storage and a programme. The storage is modelled by tapes. Our Turing machines have two tapes: an input tape and a working tape. Each tape is arbitrarily long in both directions, and they are partitioned into equally sized cells. Every cell can be empty (then it contains the *blank symbol*), or it contains exactly one symbol. There are two alphabets: one for input words and another for writing on the working tape. Each tape has a head that reads symbols and may write. On the input tape there is a read-only head, on the working tape there is a read-write head. Additionally, heads can move, and in a computation step, a head may stand still, move one cell to the left or one cell to the right. Heads move independent of each other. Furthermore, the read-only head on the input tape is only allowed to read symbols of the input word and the empty cells just to the left and to the right of the input word on the input tape. In every computation step, the Turing machine gets into a state, that represents one of a finite number of possible "inner" situations.

Instructions of Turing machines have a simple form. A situation of a Turing machine (for a fixed input) is constituted by the contents of the working tape, the machine's state and the positions of the heads on the input and working tapes. For every possible situation, a Turing machine's programme contains an instruction. An instruction can be carried out, if it is applicable in the current state and the heads read the correct symbols. Precisely, a Turing machine's instruction has the form

$$zab \longrightarrow z'chH$$

where $z$ and $z'$ denote states of the machine, $a$ is a symbol of the input alphabet, $b$ and $c$ are symbols of the working alphabet, and $h$ and $H$ represent movements of the heads on the input and working tapes, respectively. Heads can move to the left (L), to the right (R) or stand still (0). If a Turing machine is in state $z$, the head on the input tape reads symbol $a$ (which may also be the blank symbol) and the head on the working tape reads symbol $b$, then this instruction is carried out by going into state $z'$, writing symbol $c$ on the current position of the head on the working tape and moving the two heads according to $h$ and $H$.

Computations of Turing machines start with the initial standard situation: the working tape is empty, i.e., every cell contains the blank symbol, the input tape contains only blank symbols except for a finite number of consecutive cells that contain the input word. The input word does not contain blank symbols. The head on the working tape is somewhere, and the head on the input tape is on the cell containing the first symbol of the input word. If the input word is empty, the head stands on a cell containing the blank symbol. The state of the machine is a special start state. During a computation, instructions are executed as long as possible. A computation ends, if the machine is in a special end state, and it *accepts* an input if and only if the head on the input tape stands on the first input symbol and the working tape is empty.

A Turing machine is called *nondeterministic*, if it contains two instructions with the same left side. Hence, in this situation the machine can perform one of these instructions. In fact, both are performed simultaneously and independently, which results in nondeterministic computations. If there is exactly one instruction that can be performed in every situation, the Turing machine is called *deterministic*.

For random access machines and Turing machines, we define complexity measures. Let $R$ be a random access machine, and let $n$ be an input number. By $R(n)$ we denote the computation of $R$ on input $n$. Then,

$$\mathrm{time}_R(n) =_{\mathrm{def}} \min\{m : R(n) \text{ halts after } m \text{ computation steps}\}$$

is the number of computation steps that are needed by $R$ to halt on input $n$; if $R(n)$ does not end, $\mathrm{time}_R(n)$ is not defined. We restrict on random access machines with only one input parameter, since multiple input parameters can be encoded into one parameter.

**Definition 4** *Let $t$ be a function over $\mathbb{N}$. Let $R$ be a random access machine. We say that $R$ **works in time** $t$ if and only if there is a constant $m$ such that, for every number $m' \geq m$ and every input number $n$, if the binary representation of $n$ has exactly $m'$ digits then* $\mathrm{time}_R(n) \leq t(m')$.

Running time analysis of algorithms in Part III will be based on random access machines. It is clear that we will not give algorithms as random access machine programmes, and inputs will not be natural numbers. However, it is not difficult to see that this would be possible. Main obstacles are readability and comprehensibility.

For Turing machines, we define two complexity measures and distinguish between deterministic and nondeterministic computations. Let $M$ be a deterministic Turing machine, and let $x$ be an input word. By $M(x)$, we denote the computation of $M$ on input $x$. Then,

$$\mathrm{dtime}_M(x) =_{\mathrm{def}} \min\{m : M(x) \text{ halts after } m \text{ computation steps}\}$$

denotes the number of computation steps of $M$ to halt on input $x$; if $M(x)$ does not end, $\mathrm{dtime}_M(x)$ is not defined. In a similar fashion, we define the *space* that is needed by a computation. By $\mathrm{dspace}_M(x)$, we denote the number of cells of the working tape of $M$ that are visited during the computation of $M$ on $x$, if $M(x)$ ends. Otherwise, i.e., if $M(x)$ does not end, which is equivalent to $\mathrm{dtime}_M(x)$ is not defined, $\mathrm{dspace}_M(x)$ is not defined. Certainly, there are computations that do not end but visit only a finite number of cells on the working tape, but this number is not decidable for arbitrary non-ending computations. For complexity measures, we require fulfillment of Blum's axioms, that particularly require a certain decidability property [7].

**Definition 5** *Let $s$ and $t$ be functions over $\mathbb{N}$. Let $M$ be a deterministic Turing machine. We say that $M$ **works in time** $t$ if and only if there is a constant $m$ such that, for every number $m' \geq m$ and input word $x$, if the length of $x$ is equal to $m'$ then* $\mathrm{dtime}_M(x) \leq t(m')$*. Analogously, we say that $M$ **works in space** $s$ if and only if there is a constant $n$ such that, for every number $n' \geq n$ and input word $x$, if the length of $x$ is equal to $n'$ then* $\mathrm{dspace}_M(x) \leq s(n')$.

In case of nondeterministic Turing machines, the definitions of time and space complexities are more delicate. Let $M$ be a nondeterministic Turing machine, and let $x$ be an input word for $M$. By $M(x)$, we denote the computation of $M$ on input $x$. Consider $M(x)$. There may be computation paths that end, and there may be computation paths that do not end. Furthermore, some finite computation paths accept, others reject. One can define time and space for nondeterministic computations by considering only accepting computation paths. We use a stricter definition. Let

$$\mathrm{ntime}_M(x) =_{\mathrm{def}} \min\{m : \text{every computation path of } M(x) \text{ halts after } m \text{ steps}\}$$

denote the least number such that no computation path of $M(x)$ has more computation steps; if there is a non-ending computation path of $M(x)$, $\mathrm{ntime}_M(x)$ is not defined. Finally, $\mathrm{nspace}_M(x)$ denotes the least number such that no computation on any computation path of $M(x)$ visits more cells on the working tape, if all computation paths of $M(x)$ end. Otherwise, $\mathrm{nspace}_M(x)$ is not defined.

**Definition 6** *Let $s$ and $t$ be functions over $\mathbb{N}$. Let $M$ be a nondeterministic Turing machine. We say that $M$ **works in time** $t$ if and only if there is a constant $m$ such that, for every number $m' \geq m$ and input word $x$, if the length of $x$ is equal to $m'$ then* $\mathrm{ntime}_M(x) \leq t(m')$*. Analogously, we say that $M$ **works in space** $s$ if and only if there is a constant $n$ such that, for every number $n' \geq n$ and input word $x$, if the length of $x$ is equal to $n'$ then* $\mathrm{nspace}_M(x) \leq s(n')$.

Depending on the computation mode of a Turing machine, we speak of *nondeterministic time* or *deterministic space*. Similar to random access machines, Turing machines are not defined by presenting detailed programmes. Instead, we will informally describe the behaviour of a Turing machine.

An important variation of Turing machines are so-called Turing transducers. It is clear from the definition that Turing machines cannot compute functions since they are not able to produce output. The only mechanism for real world interaction is its acceptance decision. For functions, we like to have a similar computation model. A *Turing transducer* looks like an ordinary Turing machine but has an additional tape—the *output tape*. The output tape is write-only, and the head on this tape performs only one-step right moves, when a symbol is written. In one computation step, a symbol may be written or not, and this is specified in the instructions. It is clear that all complexity definitions can be transfered to Turing transducers with the only restriction that output tape space is not considered.

## 3.2 Complexity classes

In the most general sense, a *complexity class* is a set of subsets of $\Sigma^*$. Most complexity classes are defined by non-trivial syntactic or semantic language properties. And in many cases, these defining properties are based on properties of Turing machines. All complexity classes that we need are syntactic classes, and most are defined by bounding the space or time used by Turing machines for deciding or accepting languages. Let $M$ be a Turing machine. The *language decided by* $M$ is the set of input words that are accepted by $M$. (Mostly, for deciding a language it is additionally required that the machine halts on every input. This fine distinction is not necessary here.) Let $N$ be a nondeterministic Turing machine. The *language accepted by* $N$ is the set of input words $x$ for which there is an accepting computation path in $N(x)$.

**Definition 7** *Let $s$ and $t$ be functions over $\mathbb{N}$.*

*(1) The complexity class* $\mathrm{DTIME}(t)$ *is the set of languages over $\Sigma$ that can be decided by Turing machines in time $t$.*

*(2) The complexity class* $\mathrm{DSPACE}(s)$ *is the set of languages over $\Sigma$ that can be decided by Turing machines in space $s$.*

*(3) The complexity class* $\mathrm{NTIME}(t)$ *is the set of languages over $\Sigma$ that can be accepted by nondeterministic Turing machines in time $t$.*

*(4) The complexity class* $\mathrm{NSPACE}(s)$ *is the set of languages over $\Sigma$ that can be accepted by nondeterministic Turing machines in space $s$.*

Functions $t$ and $s$ are called *resource functions* since they bound resources of machines. By a combinatorial result, time- and space-bounded complexity classes can be related.

**Lemma 3.1 (folklore)**
*Let $s \geq \log$ be a resource function. Let $M$ be a nondeterministic Turing machine working in space $s$. Then, there is a constant $c > 1$ such that, for every $x \in \Sigma^*$, if there is an accepting computation path in $M(x)$ then there is an accepting computation path of length at most $2^{c \cdot s(|x|)} - 1$.*

**Proof:** Let $a$ be the number of symbols of the working alphabet of $M$, where the blank symbol is included, and let $z$ be the number of states of $M$. Let $x \in \Sigma^*$. The head on the working tape visits at most $s(|x|)$ cells during the computation $M(x)$. Hence, $M$ on input $x$ can produce at most $a^{s(|x|)}$ many configurations of the working tape, which makes $s(|x|) \cdot a^{s(|x|)}$ many situations on the working tape (the configuration of the working tape and the position of the head). The head on the input tape can visit at most $|x| + 2$ cells. So, there are only $z \cdot (|x| + 2) \cdot s(|x|) \cdot a^{s(|x|)} \leq 2^{c \cdot s(|x|)}$ many situations possible during the computation of $M$ on input $x$, where $c > 1$ depending only on $a$ and $z$.

Now, let $P = \langle S_0, \ldots, S_r \rangle$ denote a computation path of $M(x)$ where $S_r$ is the accepting end configuration. If there are $i, j \in \{0, \ldots, r\}$ such that $i < j$ and $S_i = S_j$ then $P' =_{\text{def}} \langle S_0, \ldots, S_i, S_{j+1}, \ldots, S_r \rangle$ is also a computation path of $M(x)$, that is accepting. It follows that $M(x)$ has an accepting computation path without encountering the same situation twice. By the considerations above, the statement follows.

■

We extend the definition of resource-bounded complexity classes to sets of resource functions. Let $\mathcal{C} \in \{\text{DTIME}, \text{DSPACE}, \text{NTIME}, \text{NSPACE}\}$, and let $\mathcal{R}$ be a set of resource functions. By $\mathcal{C}(\mathcal{R})$, we denote the complexity class that is the union of the complexity classes $\mathcal{C}(r)$ where $r \in \mathcal{R}$. Using these general definitions, we define concrete complexity classes. Let Pol denote the set of polynomials in one variable, and let $\mathcal{O}(\log)$ denote the set of functions of the form $f(x) = c \cdot \log x + d$ for natural numbers $c$ and $d$.

**Definition 8** *(1)* L $=_{\text{def}}$ DSPACE($\mathcal{O}$(log)) .

*(2)* NL $=_{\text{def}}$ NSPACE($\mathcal{O}$(log)) .

*(3)* P $=_{\text{def}}$ DTIME(Pol) .

*(4)* NP $=_{\text{def}}$ NTIME(Pol) .

*(5)* PSPACE $=_{\text{def}}$ DSPACE(Pol) .

*(6)* NPSPACE $=_{\text{def}}$ NSPACE(Pol) .

For a complexity class $\mathcal{C}$, the *class of complements* of $\mathcal{C}$, denoted as co$\cdot\mathcal{C}$, is the set of complements (with respect to $\Sigma^*$) of languages in $\mathcal{C}$. Famous special cases are the complement classes coNL and coNP. It is immediate that deterministic complexity classes are closed under taking complements, i.e., they are their own classes of complements. The following inclusions hold.

**Theorem 3.2 (folklore)**
L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE .

The two following equivalences are well-known but require more complicated constructions.

**Theorem 3.3 (Savitch, [78])**
PSPACE = NPSPACE .

**Theorem 3.4 (Immerman, [45]; Szelepcsényi, [80])**
NL = coNL .

The complexity class PSPACE contains problems that are really hard. But we will need complexity classes that (probably) contain even harder problems. By $2^{\mathrm{Pol}}$, we denote the set of functions of the form $2^p$ for $p$ a polynomial from Pol. Similarly, $2^{2^{\mathrm{Pol}}}$ is defined as the set of functions $2^{2^p}$ for $p$ a polynomial from Pol.

**Definition 9** *(1)* EXP $=_{\mathrm{def}}$ DTIME($2^{\mathrm{Pol}}$) .

*(2)* NEXP $=_{\mathrm{def}}$ NTIME($2^{\mathrm{Pol}}$) .

*(3)* EXPSPACE $=_{\mathrm{def}}$ DSPACE($2^{\mathrm{Pol}}$) .

*(4)* 2-NEXP $=_{\mathrm{def}}$ NTIME($2^{2^{\mathrm{Pol}}}$) .

**Theorem 3.5 (folklore)**
PSPACE $\subseteq$ EXP $\subseteq$ NEXP $\subseteq$ EXPSPACE $\subseteq$ 2-NEXP .

Similar to complexity classes, we can define function classes. Let $s$ be a resource function. By FDSPACE($s$), we denote the set of functions that can be computed by Turing transducers that work in space $s$. An important special case is the class FL, that is the set of functions that can be computed by logarithmic-space Turing transducers, hence FL = FDSPACE($\mathcal{O}(\log)$). Further function classes are defined later. Examples for concrete functions in FL are addition and multiplication of two numbers in binary representation [83]. It even holds a stronger result. By ItMult we denote the function that computes the product of a given sequence of numbers. The abbreviation stands for *iterated multiplication.*

**Theorem 3.6 (Hesse, Allender and Mix Barrington, [42])**
ItMult *is in* FL .

**Lemma 3.7 (Savitch, [78])**
*Let $s \geq \log$ and $t \geq \mathrm{id}$ be resource functions. Let $M$ be a nondeterministic Turing machine working in space $s$ and time $t$. There is a function $f \in \mathrm{FDSPACE}(s + \log t)$ such that, for every $x \in \Sigma^*$, $f(x) = (G, u, v)$ where $G$ is an acyclic graph and $u$ and $v$ are vertices of $G$ and the number of $u, v$-paths in $G$ equals the number of accepting computation paths in $M(x)$.*

**Proof:** Let $x \in \Sigma^*$, and let $\omega =_{\mathrm{def}} |x|$ be the length of $x$. Let $r =_{\mathrm{def}} s(\omega)$ and $m =_{\mathrm{def}} t(\omega)$. We will define an acyclic graph whose vertices correspond to pairs of the form $(K, n)$ where $K$ is a configuration of $M(x)$ and $n \leq m$. Every configuration describes a situation of $M(x)$ and consists of the head position on the input tape (we will alternatively use the corresponding letter of $x$), the contents of the working tape, the head position on the working tape and the machine's state. Depending on the number of states of $M$ and the size of $\Sigma$ and the working tape alphabet, there is a constant $c$ such that every configuration of $M(x)$ can be represented using $c(r+1)$ bits. So, the graph $G$ that we will construct will have $2^{c(r+1)} \cdot m$ vertices, and they can be written down using $c(r+1) + \log m$ bits.

For the set of arcs, vertices corresponding to $(K_1, n_1)$ and $(K_2, n_2)$ are adjacent if and only if $n_2 = n_1 + 1$ and one of the two cases holds

(1) situation $K_2$ can be obtained from situation $K_1$ in (exactly) one computation step

(2) $K_1 = K_2$ is the accepting end configuration.

Hence, graph $G$ can be computed by a function that uses only space $r + \log m$. By this definition, it is clear that $G$ is acyclic, and, if needed, a topological ordering of the vertices of $G$ can easily be output: first all vertices whose labels have 0 in their second component, then all vertices whose labels have 1 in their second component, and so on.

Let $\mathfrak{s}$ and $\mathfrak{a}$ denote the vertices of $G$ that correspond to the unique start and accepting end configuration of $M(x)$, respectively. In particular, the second components of the labels of $\mathfrak{s}$ and $\mathfrak{a}$ are 0 and $m$. Every $\mathfrak{s}, \mathfrak{a}$-path in $G$ corresponds to an accepting computation path of $M(x)$, and vice versa. This correspondence is 1-to-1 (a bijection), and we conclude the lemma.

■

By #L, we denote the following function class. Function $f$ is contained in #L if there is a nondeterministic logarithmic-space Turing machine $M$ working in polynomial time such that, for every $x \in \Sigma^*$, the number of accepting computation paths of $M(x)$ is equal to $f(x)$. Lemma 3.7 establishes an interesting connection between #L-functions and FL-uniform families of acyclic graphs. (This term is non-standard, but its meaning is evident with the formulation of Lemma 3.7.) Based on #L-functions, we define two further complexity classes.

**Definition 10** *Let $A$ be a set over $\Sigma$.*

*(1) $A$ is contained in* PL *if and only if there are a* #L*-function $f$ and an* FL*-function $g$ such that, for every $x \in \Sigma^*$, $x \in A \iff f(x) \geq g(x)$.*

*(2) $A$ is contained in* $\mathrm{C_=L}$ *if and only if there are a* #L*-function $f$ and an* FL*-function $g$ such that, for every $x \in \Sigma^*$, $x \in A \iff f(x) = g(x)$.*

The original definition of PL involves probabilistic Turing machines, and it is a strong result that our definition is equivalent (see [1]).

**Theorem 3.8 (Allender and Ogihara, [1]; Borodin, Cook and Pippenger, [12])**
$\mathrm{NL} \subseteq \mathrm{C_=L} \subseteq \mathrm{PL} \subseteq \mathrm{P}$.

A completely different problem class is defined at the end of this section. In case of algorithms, as we have already mentioned, complexity analysis is based on random access machines instead of Turing machines, and mostly time complexity is considered. A *linear-time algorithm* is an algorithm (which is in fact a random access machine) that works in time $c \cdot \mathrm{id}$ for some constant $c \geq 1$. Usually, this time bound is denoted as $\mathcal{O}(n)$. In case of graph algorithms, the input size is determined by the numbers of vertices and edges of the input graph, and therefore, linear time for an algorithm on graphs is denoted as $\mathcal{O}(n+m)$ for $n$ and $m$ the numbers of vertices and edges of the graph, respectively.

## 3.3 Reducibilities and complete problems

Besides the study of inclusion properties of complexity classes, the relationship between problems is one of the major topics in complexity theory. The most important relationship relates the "decision intricacies" of two problems to each other. The notion of *reducibility* is introduced, and this notion leads to the notion of complete problems for complexity classes.

A binary relation $\leq$ over the set of subsets of $\Sigma^*$ is called a *reducibility*, if it is reflexive and transitive, i.e., if $\leq$ is a quasi partial order over the power set of $\Sigma^*$. We say that $A$ *reduces* to $B$ in the sence of $\leq$, if $A \leq B$. If $A \leq B$ as well as $B \leq A$, we say that $A$ and $B$ are *isomorphic* with respect to reducibility $\leq$. Then, we also write $A \equiv B$, where $\equiv$ is an appropriate symbol. In this thesis, we apply two reducibilities, each of which uses only logarithmic space. The first one is defined straightforwardly.

**Definition 11** *Let $A, B \subseteq \Sigma^*$ be two problems. We say that $A$* ***logarithmic-space many-one reduces*** *to $B$, denoted as $A \leq_m^{\mathrm{L}} B$, if and only if there is a function $f \in \mathrm{FL}$ such that, for every $x \in \Sigma^*$, $x \in A \iff f(x) \in B$. If $A$ and $B$ are isomorphic with respect to reducibility $\leq_m^{\mathrm{L}}$, we write $A \equiv_m^{\mathrm{L}} B$.*

For concrete problems, we use reducibilities for two purposes. We can show that a given problem is among the hardest of a complexity class, and we can show that a problem is contained in a complexity class, since it is not harder than another problem of which containment in the complexity class is already known. This second application does not always work; it must be known that the reducibility is "suitable" for the involved complexity class. This notion is formalized as follows. Let $\mathcal{C}$ be a complexity class, and let $\leq$ be a reducibility. We say that $\mathcal{C}$ is *closed* under $\leq$, if, for every pair $A, B$ of subsets of $\Sigma^*$, if $B$ is contained in $\mathcal{C}$ and $A \leq B$ then $A$ is contained in $\mathcal{C}$.

**Lemma 3.9 (folklore)**
*The complexity classes* L, NL, P, NP *and* PSPACE *are closed under logarithmic-space many-one reducibility.*

In this context, the notion of complete problems naturally appears. Let $\mathcal{C}$ be a complexity class, let $\leq$ be a reducibility, and let $A$ be a problem. We say that $A$ is $\mathcal{C}$-*hard* with respect to $\leq$, if, for every set $B$ in $\mathcal{C}$, $B \leq A$. Informally spoken, $A$ is at least as hard (with respect to $\leq$) as every problem in $\mathcal{C}$. If, additionally, $A$ is contained in $\mathcal{C}$, $A$ is a $\mathcal{C}$-*complete* set with respect to $\leq$. Since hardness and completeness will always be used with respect to logarithmic-space many-one reducibility, we will shortly speak of $\mathcal{C}$-hardness and $\mathcal{C}$-completeness.

For all complexity classes that appear in this thesis, complete problems are known. For proving completeness of a problem, there are two possible ways. Let $\mathcal{C}$ be a complexity class, and let $A$ be a problem.

(1) One shows that $A$ is contained in $\mathcal{C}$ and that there is a reducing function—a *reduction*—from every problem in $\mathcal{C}$ to $A$, i.e., showing $\mathcal{C}$-hardness of $A$.

(2) One shows that $A$ is contained in $\mathcal{C}$ and that there is a problem $B$ that is known to be $\mathcal{C}$-hard and that reduces to $A$. $\mathcal{C}$-hardness of $A$ follows by transitivity of the involved reducibility.

It is clear that method (2) can only be applied if a hard problem is already known. Method (1) is carried out by defining so-called *master reductions*, which are reductions that are usually based on computations.

For the complexity class NL, we give three (or five) complete problems, that are based on graphs. The *graph accessibility problem* GAP is the set of triples $(G, u, v)$ where $G$ is a directed graph and contains a $u, v$-path. The inverse problem, the *graph inaccessibility problem* GIP, is the set of triples $(G, u, v)$ where $G$ is a directed graph and does not contain a $u, v$-path. Remember that existence of paths in directed graphs is not equal to connectivity questions. Finally, the *acyclicness problem* ACYC is the set of acyclic (directed) graphs.

**Theorem 3.10 (Savitch, [78])**
GAP *is* NL-*complete.*

**Proof:** Let $(G, u, v)$ be an instance of GAP. Let $n$ be the number of vertices of $G$. For determining whether $G$ contains a $u, v$-path, guess, starting at vertex $u$, a sequence $(x_1, \ldots, x_n)$ of $n$ vertices such that there is an arc from $x_i$ to $x_{i+1}$ for every $i \in \{1, \ldots, n-1\}$. Accept if and only if the sequence contains $v$. Since the length of the input is at least $n$, two consecutive vertices of the guessed sequence can be written on the working tape using only logarithmic space, and adjacency can be checked also using only logarithmic space. Hence, GAP is in NL.

For hardness, let $A \in$ NL, and let $M$ be a nondeterministic Turing machine accepting $A$ in logarithmic space. Due to Lemma 3.1, there is a polynomial $p$ such that, for every $x \in \Sigma^*$, $M(x)$ contains an accepting computation path if and only if $M(x)$ contains an accepting computation path of length at most $p(|x|)$. Applying Lemma 3.7, there is an FL-function realising the reduction from $A$ to GAP.

■

**Corollary 3.11**
GIP *is* NL*-complete.*

**Proof:** We show that GIP is contained in coNL. Let $(G, u, v)$ be an instance of GIP. The algorithm of Theorem 3.10 showing GAP $\in$ NL can be used, and we obtain GIP $\in$ coNL. Analogous to the reduction in the proof of Theorem 3.10, GIP is coNL-complete, and hence NL-complete applying Theorem 3.4.

■

We only remark that GAP and GIP are NL-complete even if they are restricted to acyclic graphs, as the proof of Theorem 3.10 shows, since it is based on the construction of an acyclic graph in Lemma 3.7. These two problems are denoted as ACYCGAP and ACYCGIP, and we will use them later for proving hardness results.

**Theorem 3.12** ACYC *is* NL*-complete.*

**Proof:** Let $G$ be an instance of ACYC. Let $n$ denote the number of vertices of $G$. It holds that $G$ is not acyclic if and only if there is a sequence $(x_1, \ldots, x_{n+1})$ of $n+1$ vertices of $G$ such that $(x_i, x_{i+1})$ is an arc of $G$ for every $i \in \{1, \ldots, n\}$. Note that, if such a sequence exists, it must contain a vertex twice, and $G$ has a cycle. Reusing the algorithm of Theorem 3.10, such a sequence can be found in logarithmic space, and ACYC $\in$ coNL, i.e., ACYC $\in$ NL due to Theorem 3.4.

For hardness of ACYC, let $A \in$ NL. Let $M$ be a Turing machine accepting the complement of $A$ using only logarithmic space. Such a machine exists due to Theorem 3.4. Application of Lemmata 3.1 and 3.7 shows the existence of an FL-function that outputs, on input $x$, an acyclic graph $G$ and vertices $u$ and $v$ such that $G$ has a $u, v$-path if and only if $M(x)$ has an accepting computation path, if and only if $x$ does not belong to $A$. Let $G'$ emerge from $G$ by adding arc $(v, u)$, that does not belong to $G$. Then, $G'$ is acyclic if and only if $G'$ does not have a $u, v$-path, if and only if $G$ does not have a $u, v$-path, if and only if $x \in A$. The described reduction can be performed by an FL-function.

■

More complete problems, especially for the complexity classes NP and PSPACE, will be defined in the following chapters, where they are needed. Most of the problems defined in Part II turn out to be complete for appropriate complexity classes.

For the second reducibility, we need functions that can be computed by nondeterministic logarithmic-space Turing transducers. Functions computed by nondeterministic Turing transducers, however, always cause confusion, that is why the nondeterminism is shifted to an oracle. So, we begin by defining relativized (nondeterministic) computations. An *oracle Turing machine* $M$ with oracle set $A$ is a Turing machine that additionally has a third tape, the *oracle* tape, on which *queries* to set $A$ are written. The query tape is write-only, and the head of this tape simply runs from left to right (like for the output tape of Turing transducers). For interaction, there are three further states of the machine besides the usual states: asking a query, query is positively answered, query is negatively answered. If, during a computation of $M$, the word on the query tape shall be asked, $M$ goes into the query state. In the next step, the query is answered: if $q$ is the word on the query tape, $q \in A$ is the question. $M$ goes into the appropriate answer state, clears the oracle tape and continues its work. A crucial point is the computation mode when generating oracle queries. According to the rule introduced by Ruzzo, Simon, Tompa, the machine is required to work deterministically when generating a query, i.e., from the step writing the first symbol of the oracle query on the query tape to the step when the query is asked [77]. This restriction is only of importance for oracle Turing machines that have only a small working tape, i.e., if the query word cannot be generated first on the working tape and then simply copied to the query tape. The time complexity of an oracle Turing machine is defined as usual by counting the number of computation steps. For the space complexity of oracle Turing machines only the working tape is taken into account. Oracle Turing transducers are like oracle Turing machines, that additionally have a one-way write-only output tape.

So, by the definition of oracle Turing transducers, it is clear how a (deterministic) logarithmic-space oracle Turing transducer works: on its working tape only logarithmically many cells are visited during a computation, and the lengths of queries do not count.

**Definition 12** *Let $f$ be a function. We say that $f \in \mathrm{FNL}$ if and only if there is a set $A \in \mathrm{NL}$ and a logarithmic-space oracle Turing transducer $M$ such that $M$ with oracle $A$ computes $f$.*

The definition of the function class FNL is motivated by an inclusion result for NL. A set $B$ is in $\mathrm{L}^{\mathrm{NL}}$ if there is a set $A \in \mathrm{NL}$ and a logarithmic-space oracle Turing machine $M$ such that $M$ with oracle $A$ decides $B$. Analogously, the complexity class $\mathrm{L}^{\mathrm{P}}$ is defined, that allows more powerful oracle sets. The complexity classes $\mathrm{L}^{\mathrm{NL}}$ and $\mathrm{L}^{\mathrm{P}}$ are also called the *logarithmic-space Turing closures* of NL and P, respectively. By $\mathrm{NL}^{\mathrm{NL}}$, we denote the set of problems $B$ for which there are a nondeterministic logarithmic-space oracle Turing machine $M$ and a set $A \in \mathrm{NL}$ such that $B$ is accepted by $M$ with oracle $A$.

**Theorem 3.13** $\mathrm{L}^{\mathrm{NL}} \subseteq \mathrm{NL}^{\mathrm{NL}} \subseteq \mathrm{NL} \subseteq \mathrm{L}^{\mathrm{P}} \subseteq \mathrm{P}$.

**Proof:** The first and third inclusions are clear, so that it suffices to show the second and fourth inclusions. We begin with the second inclusion. Let $B \in \mathrm{NL}^{\mathrm{NL}}$. Then, there is a set $A \in \mathrm{NL}$ and a nondeterministic logarithmic-space oracle Turing machine $M$ such that $M$ with oracle $A$ accepts $B$. We construct a nondeterministic logarithmic-space Turing machine $M'$ that accepts $B$. Let $N_1$ and $N_2$ be nondeterministic logarithmic-space Turing machines accepting $A$ and $\overline{A}$, respectively. Remember that $N_2$ exists due to Theorem 3.4. On input $x$, $M'$ works as follows: $M'$ basically simulates the work of $M$ on input $x$. Query generation is ignored. When

$M(x)$ asks a query, $M'(x)$ guesses whether the answer is 'yes' or 'no' and accordingly simulates $N_1$ or $N_2$. However, the query word has not yet been generated. Whenever $N_1$ or $N_2$ wants to read a symbol of the query word, $M'$ generates this symbol. This can be done in logarithmic space, since $M(x)$ works only in logarithmic space. Note that the simulation cannot always begin with the start situation, since query generation may be adaptive, i.e., it may depend on previous queries. Though, it suffices to simulate query generation from the beginning of the actual query generation of $M(x)$. The corresponding configuration can be saved in logarithmic space. It is clear that either the simulation of $N_1$ or the simulation of $N_2$ ends with acception, and the simulation of $M(x)$ is continued only on accepting computation paths. In total, only logarithmic space is used, hence $M'$ is a nondeterministic logarithmic-space Turing machine accepting $B$.

The inclusion $\mathrm{L}^{\mathrm{P}} \subseteq \mathrm{P}$ is rather easy. Note that a logarithmic-space oracle Turing machine is also a polynomial-time machine. Let $B \in \mathrm{L}^{\mathrm{P}}$. Let $M$ be a logarithmic-space oracle Turing machine and let $A \in \mathrm{P}$ such that $M$ with oracle $A$ accepts $B$. Let $N$ be a polynomial-time Turing machine deciding $A$. We construct a polynomial-time Turing machine $M'$ that decides $B$. On input $x$, let $M'$ simulate $M(x)$, where oracle queries are written in a separate area of the working tape. Whenever a query is asked to the oracle by $M(x)$, $M'$ interrupts the simulation of $M(x)$ and simulates the work of $N$ on the query. Since $M(x)$ can generate at most polynomially many queries, since every query is of size polynomial in the length of $x$ and $N$ is a polynomial-time Turing machine, the overall simulation, i.e., the computation of $M'(x)$, takes only polynomial time. Hence, $M'$ is indeed a polynomial-time Turing machine, and $B \in \mathrm{P}$.

■

This proof also shows that FNL-functions can be computed by nondeterministic logarithmic-space Turing transducers. It is clear that there may be numerous computation paths that produce an output, but they all produce the same output. The function class FNL was considered by Àlvarez, Balcázar, Jenner, and they gave further characterisations of this class [3].

By using FNL, we define a nondeterministic analogue of logarithmic-space many-one reducibility.

**Definition 13** *Let $A, B \subseteq \Sigma^*$ be two problems. We say that $A$* ***nondeterministic logarithmic-space many-one reduces*** *to $B$, denoted as $A \leq_m^{\mathrm{NL}} B$, if and only if there is a function $f \in \mathrm{FNL}$ such that, for every $x \in \Sigma^*$, $x \in A \iff f(x) \in B$.*

**Lemma 3.14** NL *is closed under nondeterministic logarithmic-space many-one reducibility.*

**Proof:** Let $A \in \mathrm{NL}$, and let $B$ be a subset of $\Sigma^*$. Assume that $B \leq_m^{\mathrm{NL}} A$ via reducing function $f$. By the proof of Theorem 3.13, there is a nondeterministic logarithmic-space Turing transducer $T$ computing $f$. We define a nondeterministic logarithmic-space oracle Turing machine $M$ as follows: $M$ on input $x$ simulates $T(x)$, and instead of writing the output of $T(x)$ on the output tape, $M$ writes it on the query tape. In the last but second step, $M$ asks the query, and it accepts if and only if the query is positively answered. Using $A$ as oracle set, $B$ has been shown to be in $\mathrm{NL}^{\mathrm{NL}}$, which is equal to NL due to Theorem 3.13.

■

# Part II

# Finite recurrent systems

*Regular languages over a fixed alphabet are sets of words that are inductively defined: The empty set and singleton sets containing only letters from the alphabet are regular languages. If we have two regular languages, their union and concatenation are regular languages as well as Kleene iteration of regular languages. Such descriptions are called* regular expressions. *It is a well-known fact that regular languages are just the languages that can be accepted by finite automata. To get an idea finite automata can be considered Turing machines that use no space on the working tape.*

*Stockmeyer and Meyer were interested in the computational complexity of the problem, given a regular expression and a word, is the word contained in the language described by the expression. It turned out that this problem is hard and becomes even harder if further operations are allowed such as squaring, which is an abbreviation for concatenating a language with itself [79]. Inspired by this type of problems, they defined further so-called membership problems for other environments. In the context of our considerations, the most influencial definitions were membership problems for integer expressions, that are expressions in the style of regular expressions where union, addition and complementation (with respect to the set of natural numbers) are applied to initially singleton sets containing only natural numbers. Integer expressions* *represent sets of natural numbers. By the way, addition of two sets of natural numbers is defined elementwise.*

*Based on the work of Stockmeyer and Meyer, McKenzie and Wagner broadened the sets of natural numbers for which membership of a given number shall be decided. Instead of defining sets by integer expressions, they used arithmetical circuits which provide an even more succinct representation of sets of natural numbers with respect to integer expressions. And a second aspect was modified. Besides the already involved operations union, addition and complementation, also intersection and multiplication of sets were considered. McKenzie and Wagner investigated the complexity of the membership problem, given a natural number and an arithmetical circuit whose input vertices are labelled with natural numbers and whose inner vertices are attached labels from the set of the five operations, whether the number is contained in the set defined by the circuit [61]. The authors distinguished a number of cases by restricting the set of operations vertices of circuits are labelled with, and they obtained completeness results for well-known complexity classes such as* NL, NP *and* PSPACE.

*In this series, our work, that is presented in this part of the thesis, extends the previous definitions and questions. We mainly consider sets of natural numbers that are generated by iterated application of functions to sets of natural numbers. The mathematical structure that formally defines this process is called a* recurrent system. *Recurrent systems are inspired by usual recurrences. For recurrent systems, i.e., for sets of natural numbers that are represented by recurrent systems, we investigate the complexity of two types of problems: given a natural number and a recurrent system, is the number contained in the set that is generated by the system after application of a specified number of iteration steps, and is the given number contained in some set that is generated after a finite number of iteration steps? The second question may be understood as the question whether a given number can be generated by the given recurrent system. We do not consider general recurrent systems but restrict the set of possible operations and study the complexities of these restricted problems. Similar to the work of McKenzie and Wagner, we will also obtain completeness results for the complexity classes* NL, NP *and* PSPACE.

*The study of the computational complexity of the membership problems inspired the definition of further problems that appear as auxiliary problems for determining containment or hardness of a membership problem. One of these problems asks for numbers of walks in arbitrary directed graphs. To obtain a decision problem definition, we consider two variants: is the number of walks of given length from a vertex to another vertex bounded below by a given number, or is it equal to a given number. The complexity of such problems is studied, where we distinguish between unary and binary representation of the queried numbers of walks.*

# Chapter 4
# The complexity of auxiliary problems

We will study the complexity of two types of problems: first, we will investigate special number-of-walks problems, second, we will consider a problem asking for a solution of a set of equations. Our number-of-walks problems basically ask whether a given directed graph has at least a given number of walks of given length from one to another vertex. These problems fit well into a series of past work. At the beginning, the graph accessibility problem can be seen, that asks whether a given directed graph has at least one walk from one to another vertex. Note that a length is not specified. An analogous question can be asked for undirected graphs, and we obtain the connectivity problem. Recently, Reingold showed that the connectivity problem is very easy with respect to our complexity measures, in fact it is decidable in logarithmic space, i.e., contained in L [72].

A number of authors studied further problems asking for special walks in graphs. An early result is by Valiant. He investigated the complexity of computing the number of "self-avoiding paths" from one to another specified vertex in directed as well as undirected graphs. Valiant's notion of self-avoiding paths corresponds to our definition of paths: a self-avoiding path, or self-avoiding walk with our definitions, is a walk that visits a vertex at most once. Valiant showed that his problems both are #P-complete with respect to an appropriate reducibility [82], that is based on our $\leq_m^{\mathrm{L}}$-reducibility. Liśkiewicz, Ogihara, Toda restricted Valiant's problems to special graphs and asked for paths (self-avoiding walks) of a specified length. They also obtained #P-completeness results [57]. Finally, Allender, Reinhardt, Zhou considered acyclic graphs and asked for the number of paths from one to another specified vertex. Since walks in acyclic graphs contain every vertex at most once, they are already paths. The authors showed that asking a lower bound given in unary representation for the number of paths is NL-complete [2]. Our number-of-walks problems consider walks of given length but arbitrary directed graphs.

The second type of problems, asking for a solution of a set of congruence equations, can be thought inspired by the Chinese Remainder Theorem. This fundamental result shows that all congruence equations of a system of congruence equations can simultaneously be satisfied, if the moduli are pairwise relatively prime. If we relax the requirement of relatively prime moduli, the existence of a solution is not guaranteed. To furtherly complicate our problem, we allow several possible residues for every module. The complexity of this and an interesting related problem will be determined.

## 4.1 Number-of-walks and power-of-matrix problems

Accessibility in graphs has been studied for a long time. The most popular problem is the graph accessibility problem, that asks for the existence of a path in a directed graph from one to another given vertex. Allender, Reinhardt, Zhou restricted the class of considered graphs but extended the problem: they asked for the number of paths from one to another given vertex of an acyclic graph [2]. We extend this last mentioned problem with respect to two aspects: we consider arbitrary directed graphs and ask for the number of walks. Let $G = (V, A)$ be a directed graph, and let $a$ and $b$ be vertices of $G$. An *$a, b$-walk* of length $k$ in $G$ is a sequence $(x_0, \dots, x_k)$ of vertices of $G$ where $x_0 = a$ and $x_k = b$ and $(x_i, x_{i+1}) \in A$ for every $i \in \{0, \dots, k-1\}$. Note that vertices may have multiple occurrences in the sequence. Two $a, b$-walks are equal if and only if they coincide in every position.

Problem definitions

NoW($\alpha$) for $\alpha \in \{1, 2\}$ (number of walks in a graph).
**Instance** $(G, k, \nu, u, v)$ where $G = (V, A)$ is a simple finite directed graph, $u, v \in V$ and $k, \nu \in \mathbb{N}$, and $k$ is given in binary representation and $\nu$ is given in $\alpha$-ary representation.
**Question** Are there $\nu$ $u, v$-walks of length $k$ in $G$?

NoMW($\alpha$) for $\alpha \in \{1, 2\}$ (number of walks in a marked graph).
**Instance** $(G, M, k, \nu, u, v)$ where $G = (V, A)$ is a simple finite directed graph, $M \subseteq V$, $u, v \in V$ and $k, \nu \in \mathbb{N}$, and $k$ is given in binary representation and $\nu$ is given in $\alpha$-ary representation.
**Question** Are there $\nu$ $u, v$-walks in $G$ each of which containing exactly $k$ vertices from set $M$?

Let $G = (V, A)$ be a directed graph, and let $M \subseteq V$ be the set of marked vertices of $G$. If $W$ is a walk in $G$ containing exactly $k$ marked vertices, we say that $W$ is a *$k$-marked* walk in $G$. From the above problems we derive their *exact* variants xNoW($\alpha$) and xNoMW($\alpha$) in the following way: xNoW($\alpha$) is the set of tuples $(G, k, \nu, u, v)$ from NoW($\alpha$) where there are exactly $\nu$ $u, v$-walks of length $k$ in $G$; a similar definition holds for xNoMW($\alpha$).

At first, we show some interesting relations between these problems. Let $h$ be an $n$-ary Boolean function. We say that *$A$ logarithmic-space truth-table reduces to $B$ via $h$*, denoted as $A \leq^{\mathrm{L}}_{h\mathrm{tt}} B$, if there is an FL-function that, on input $x$, outputs $n$ words $y_1, \dots, y_n$ such that $x \in A$ if and only if $h(c_B(y_1), \dots, c_B(y_n)) = 1$. Observe that, for every Boolean function, NL and P are closed under fixed-function logarithmic-space truth-table reducibility. (Fixed-function logarithmic-space truth-table reducibility is a special case of logarithmic-space Turing reducibility, under which NL and P are closed due to Theorem 3.13.)

**Lemma 4.1** *Let $a(x_1, x_2) =_{\mathrm{def}} (x_1 \wedge \neg x_2)$. For $\alpha \in \{1, 2\}$,*

*(1)* xNoW($\alpha$) $\leq^{\mathrm{L}}_{a\mathrm{tt}}$ NoW($\alpha$).

*(2)* xNoMW($\alpha$) $\leq^{\mathrm{L}}_{a\mathrm{tt}}$ NoMW($\alpha$).

**Proof:** We prove statement (1) of the lemma; the proof of statement (2) is analogous. Let $(G,k,\nu,u,v)$ be an instance of $\mathrm{xNoW}(\alpha)$. $(G,k,\nu,u,v) \in \mathrm{xNoW}(\alpha)$ if and only if $(G,k,\nu,u,v) \in \mathrm{NoW}(\alpha)$ and $(G,k,\nu+1,u,v) \notin \mathrm{NoW}(\alpha)$. Adding a constant to a number can be done in logarithmic space for unary as well as binary number system. ■

A second, at first glance more surprising result relates number-of-walks problems for marked and unmarked graphs. There is an immediate direction, that we prove first.

**Lemma 4.2** *Let* $\alpha \in \{1,2\}$.

*(1)* $\mathrm{NoW}(\alpha) \leq^{\mathrm{L}}_{m} \mathrm{NoMW}(\alpha)$.

*(2)* $\mathrm{xNoW}(\alpha) \leq^{\mathrm{L}}_{m} \mathrm{xNoMW}(\alpha)$.

**Proof:** It suffices to prove statement (1); statement (2) is proved completely analogous. Let $(G,k,\nu,u,v)$ be an instance of $\mathrm{NoW}(\alpha)$. Let $V$ denote the set of vertices of $G$. It holds that $W$ is a $u,v$-walk of length $k$ if and only if it is a $u,v$-walk containing exactly $k+1$ vertices. Hence, $(G,k,\nu,u,v) \in \mathrm{NoW}(\alpha)$ if and only if $(G,V,k+1,\nu,u,v) \in \mathrm{NoMW}(\alpha)$. ■

The other direction, from the marked to the unmarked problem version, needs a more sophisticated construction. It can be described as follows. It is determined whether the number of walks is infinite, and if not a new graph that preserves the number of walks and without marked vertices is constructed. To achieve this, we have to decide whether a graph contains a certain walk of given length. Let $G = (V,A)$ be a directed graph. For two vertices $u,v \in V$ and a number $k \in \mathbb{N}$, let $W_G$ be the function such that $W_G(\overrightarrow{uv},k) \in \{0,1\}$ and $W_G(\overrightarrow{uv},k) = 1$ if and only if there is a $u,v$-walk of length $k$ in $G$. With $W_G$, we associate the decision problem WALK that is the set of tuples $(G,u,v,k)$ where $G=(V,A)$ is a directed graph, $u,v \in V$, $k \in \mathbb{N}$ is given in binary representation and $W_G(\overrightarrow{uv},k) = 1$. For an acyclic single-source graph $G=(V,A)$ we define the *depth* of a vertex $u$ of $G$ as the length of a longest path from the source vertex to $u$ in $G$. Note that acyclic single-source graphs are connected. By DEPTH we denote the set of triples $(G,u,d)$ where $G=(V,A)$ is an acyclic single-source graph, and $u \in V$ has depth $d$ in $G$.

**Lemma 4.3** WALK *and* DEPTH *are in* NL.

**Proof:** For the complexity of WALK, let $(G,u,v,k)$ be an instance of WALK. Let $W = (u_0,\ldots,u_k)$ be a $u,v$-walk in $G$. If $k \leq (n+1)^2$ we can find a walk straightforwardly by starting with $u$ and guessing $k$ consecutive vertices. Now, let $k > (n+1)^2$. Let $r \in \{0,\ldots,n-1\}$ such that there is $d \leq n-r$ and $u_r = u_{r+d}$. Consider the vertex sequence $(u_r, u_{r+d}, u_{r+2d}, \ldots, u_{r+ed})$ where $k-d < r+ed \leq k$. Note that $e > n$ and that there are $i,j \in \{1,\ldots,e\}$, $i<j$, such that $u_{r+id} = u_{r+jd}$.

Then, $(u_0,\dots,u_{r+d} = u_r,\dots,u_{r+d} = u_r,\dots,u_{r+id},u_{r+jd+1},\dots,u_k)$ is a $u,v$-walk of length $k$ in $G$. Iterated application of this argument shows that there is a $u,v$-walk $W' = (v_0,\dots,v_k)$ where $u_i = v_i$ for all $i \in \{0,\dots,r+d\}$ and $v_{r+jd} = u_r$ for all $j \in \{0,\dots,e-n+1\}$. This characterisation gives a nondeterministic logarithmic-space algorithm to solve the problem. First, guess vertex $w$ of $G$ and $d \le n$ and verify the existence of a walk $(w_0,\dots,w_d)$ where $w_0 = w_d = w$. Then, verify the existence of a walk $(x_0,\dots,x_m)$ where $m < n + (n-1)\cdot d + d = n\cdot(d+1)$, $m \equiv k \pmod{d}$, and that there is $i < n$ such that $x_i = w$.

For the complexity of Depth, let $(G,u,d)$ be an instance of Depth. In nondeterministic logarithmic space, it can be determined by standard procedures whether $G$ is a connected acyclic single-source graph. Let $s$ be the unique source vertex of $G$. Determine the largest number $k$ such that $W_G(\overrightarrow{su},k) = 1$, which can be done in logarithmic space with oracle access to Walk. If the largest number is equal to $d$ accept, otherwise reject. This gives a nondeterministic logarithmic-space algorithm due to Theorem 3.13.

■

**Theorem 4.4** *Let* $\alpha \in \{1,2\}$.

*(1)* $\text{NoMW}(\alpha) \le_m^{\text{NL}} \text{NoW}(\alpha)$.

*(2)* $\text{xNoMW}(\alpha) \le_m^{\text{NL}} \text{xNoW}(\alpha)$.

**Proof:** We first prove statement (1) of the theorem; the proof of statement (2) is discussed at the end. Let $(G,M,k,\nu,u,v)$ be an instance of $\text{NoMW}(\alpha)$. Let $n =_{\text{def}} |V|$ be the number of vertices of $G$. If there is a vertex $x \in V$ such that $(x,x) \in A$, replace this loop by the walk $(x,w,x)$ for some new vertex $w$. If $u \notin M$ then add a new vertex $u'$ to $G$ that is adjacent to every successor of $u$, add $u'$ to $M$ and increase $k$ by 1; $u'$ becomes the new start vertex. If $v \notin M$ then add a new vertex $v'$ to $G$ that is successor of every predecessor of $v$, add $v'$ to $M$ and increase $k$ by 1; $v'$ becomes the new end vertex. So, without loss of generality we assume that $G$ is loop-free and $u,v \in M$.

[$G$ contains infinitely many $k$-marked $u,v$-walks.]

For every $x \in M$, let $G_x =_{\text{def}} G \setminus (M \setminus \{x\})$. In other words, $G_x$ is the subgraph of $G$ where all vertices from $M$ except for $x$ are deleted. For every pair $x,z \in M$ of marked vertices, let $K'_{xz}$ denote the set of vertices $w \in V$ for which there exist an $x,w$-path in $G_x$ and a $w,z$-path in $G_z$. In other words, $K'_{xz}$ is the set of unmarked vertices of $G$ that appear on 2-marked $x,z$-walks in $G$. Let $K_{xz} =_{\text{def}} K'_{xz} \cup \{x,z\}$, if $K'_{xz} \ne \emptyset$ or $(x,z) \in A$; otherwise let $K_{xz} =_{\text{def}} K'_{xz} = \emptyset$. With oracle set Walk it can be tested whether $K_{xz}$ is empty. Suppose there are $x,z \in M$ such that $G[K'_{xz}]$ has a cycle. Then, there are infinitely many 2-marked $x,z$-walks in $G$. Such vertices $x$ and $z$ can be found in logarithmic space with oracle access to Walk. If there are a $k_1$-marked $u,x$-walk and a $k_2$-marked $z,v$-walk such that $k_1 + k_2 = k$ then the input tuple must be accepted. To verify the existence of these walks we construct graph $G^*$ which contains a copy of every vertex in $M$ and arc $(p,q)$ for two vertices $p,q \in M$ if and only if $K_{pq}$ is not empty. For $p,q \in M$ where $K_{pq} \neq \emptyset$, let $G'_{pq}$ be the disjoint union of two copies of $G^*$ where the vertices of the first copy have index 1 whereas the vertices of the second copy have index 2 and that contains the arc $(p_1, q_2)$. Then, $G$ contains a $k$-marked $u,v$-walk containing $p$ and $q$ as consecutive vertices from $M$ if and only if $G'_{pq}$ contains a $u_1,v_2$-walk of length $k$. For every pair of vertices $x,z \in M$ where $G[K'_{xz}]$ contains a cycle verify whether $G'_{xz}$ has a $u_1,v_2$-walk of length $k$. This can be done using WALK as oracle. If such a walk has been found output some element of NOW$(\alpha)$.

[$G$ contains only finitely many $k$-marked $u,v$-walks.]

Let it be the case that no such walk exists (and has been found). For every pair $x,z$ of marked vertices of $G$, let $G_{xz}$ emerge from $G[K_{xz}]$ where, in case of $x \neq z$, arcs of the forms $(w,x)$ and $(z,w)$ are deleted. If $x \neq z$ and $G_{xz}$ is not acyclic, let $G_{xz}$ be empty. If $G_{xx} \setminus (V \times \{x\})$ is not acyclic, let $G_{xx}$ be empty. For every pair of vertices $x,z \in M$, $x \neq z$, determine the depth of $z$ in $G_{xz}$. Remember that, by construction, $G_{xz}$ is connected, acyclic and single-source (if not empty). Let the maximum value be denoted by $a'$. For every $x \in M$, determine the depth of every predecessor of $x$ in $G_{xx}$ in $G_{xx} \setminus (V \times \{x\})$. Let the maximum be denoted by $a''$. Let $a =_{\text{def}} \max\{a', a''+1\}$. With oracle access to DEPTH, $a$ can be determined in logarithmic space. Observe the following: Let $W = (w_0, \dots, w_\ell)$ be a $k$-marked $u,v$-walk in $G$, and let $I \subseteq \{0, \dots, \ell\}$ such that, for all $i \in \{0, \dots, \ell\}$, $w_i \in M$ if and only if $i \in I$. Then, for consecutive $w_i$, $w_j$, $i,j \in I$, $(w_i, w_{i+1}, \dots, w_j)$ is a walk in $G_{w_i w_j}$ (remember that the first case, considered in the paragraph above, has not been applicable). We add new vertices to make walks between vertices from $M$ of equal length. Let $x,z \in M$. Let $H_{xz}$ emerge from $G_{xz}$ as follows. $H_{xz}$ contains a copy of every vertex in $G_{xz}$. (We add further vertices and arcs.) For every pair of vertices $p$ and $q$ where $(p,q)$ is an arc in $G_{xz}$, if $x \neq z$ determine the depths of $p$ and $q$ in $G_{xz}$, if $x = z$ and $q \neq x$ determine the depths of $p$ and $q$ in $G_{xx} \setminus (V \times \{x\})$, if $x = q = z$ determine the depth of $p$ in $G_{xx} \setminus (V \times \{x\})$. Let the depths be numbers $d_p$ and $d_q$, respectively. If $q \neq z$, let $d =_{\text{def}} d_q - d_p - 1$; if $q = z$, let $d =_{\text{def}} a - d_p - 1$. Let $y_1, \dots, y_d$ be new vertices. Add walk $(p, y_1, \dots, y_d, q)$ to $H_{xz}$. Observe that there is a 1-to-1 correspondence between the $x,z$-walks in $G_{xz}$ and $H_{xz}$. Let $H$ be the union of all $H_{xz}$ for $x,z \in M$ where all copies of the same vertex from $M$ are glued together. Note that $H$ is a graph similar to $G^*$, that still contains information about the number of walks. It holds that the $k$-marked $u,v$-walks in $G$ correspond to the $u,v$-walks of length $a \cdot (k-1)$ in $H$ and vice versa. $H$ can be computed in logarithmic space using oracle sets WALK and DEPTH. Hence, $(G, M, k, \nu, u, v) \in$ NOMW$(\alpha)$ if and only if $(H, a \cdot (k-1), \nu, u, v) \in$ NOW$(\alpha)$. Note that $a \cdot (k-1)$ can be computed in logarithmic space, since multiplication is an FL-function. It is even possible to choose $a$ as a power of 2, which makes multiplication trivial. We conclude statement (1) by Lemma 4.3.

For statement (2), it suffices to observe that, if there are infinitely many $k$-marked $u,v$-walks, the input must be rejected. The remaining reduction is analogous to the construction above.

■

We have seen that it basically suffices to consider only the problems for unmarked graphs. These problems are also related to problems for powers of matrices.

Problem definition

POM($\alpha$) for $\alpha \in \{1,2\}$ (element of a power of a matrix).
**Instance** $(M,k,a,i,j)$ where there is $n \geq 1$ and $M$ is an $n \times n$ matrix over $\mathbb{N}$, $k,a \geq 0$, $i,j \in \{1,\dots,n\}$. All entries of $M$ and $i$, $j$ and $k$ are given in binary representation, and $a$ is given in $\alpha$-ary representation.
**Question** $(M^k)_{ij} \geq a$?

Similar to the number-of-walks problems, we also consider an exact variant of POM($\alpha$), that is denoted as xPOM($\alpha$) and obtained by simply replacing '$\geq$' in the problem definition of POM($\alpha$) by '$=$'. The relations of POM($\alpha$) and xPOM($\alpha$) to NoW($\alpha$) and xNoW($\alpha$), respectively, are based on a result from graph theory. Let $G = (V,A)$ be a directed graph. Let $n$ be the number of vertices of $G$. By $\mathcal{A}dj(G)$ we denote the $n \times n$ matrix $M$ over $\{0,1\}$ for which holds $M_{ij} = 1$ if and only if $(i,j) \in A$ for $i,j \in \{1,\dots,n\}$. $\mathcal{A}dj(G)$ is called the *adjacency matrix* of $G$.

**Lemma 4.5 (folklore)**
*Let $G = (V,A)$ be a directed graph. Let $n =_{\text{def}} |V|$ be the number of vertices of $G$, and let $k \geq 1$ and $i,j \in \{1,\dots,n\}$. Let $M =_{\text{def}} \mathcal{A}dj(G)$ be the adjacency matrix of $G$. Then, $(M^k)_{ij}$ is the number of $i,j$-walks of length $k$ in $G$.*

**Proof:** We prove the statement by induction over $k$. For $k = 0$, $M^k = M^0$ is the matrix containing 1's only on the main diagonal, and the statement holds. Now, let it be true that $(M^k)_{ij}$ is the number of $i,j$-walks of length $k$ in $G$. Then, the number of $i,j$-walks of length $k+1$ in $G$ is the sum of the numbers of $i,j'$-walks of length $k$ in $G$ for all predecessors $j'$ of $j$. So, the statement follows by the definition of the product $M^k \cdot M$.

■

**Lemma 4.6** *Let $\alpha \in \{1,2\}$.*

*(1)* NoW($\alpha$) $\equiv^{\text{L}}_m$ POM($\alpha$).

*(2)* xNoW($\alpha$) $\equiv^{\text{L}}_m$ xPOM($\alpha$).

**Proof:** We only prove statement (1); the proof of statement (2) is analogous. The problem NoW($\alpha$) reduces to POM($\alpha$) due to Lemma 4.5 by computing matrix $\mathcal{A}dj(G)$, which is possible in logarithmic space. So, let $(G,k,\nu,u,v)$ be an instance of $\mathrm{NoW}(\alpha)$. Then, $(G,k,\nu,u,v)$ in $\mathrm{NoW}(\alpha)$ if and only if $(\mathcal{A}dj(G),k,\nu,u,v)$ in $\mathrm{POM}(\alpha)$. Remember that vertices $u$ and $v$ are natural numbers.

For the converse reduction, let $(M,k,a,i,j)$ be an instance of $\mathrm{POM}(\alpha)$, and let $\omega$ be the size of this instance. Consider the acyclic graph depicted in Figure 3. All $u,v$- and $u,w$-walks have the same length and there are exactly $1+8=9$ $u,v$-walks and $16+64=80$ $u,w$-walks. Graphs of the form of $Z_6$, as it is denoted in Figure 3, play the central part in our reduction. A formal definition is as follows. By $Z_0$ we mean a single vertex, and $Z_s$ is extended to $Z_{s+1}$ by glueing together the sink of $Z_s$ with the source vertex 1 of the graph $Z_1 =_{\mathrm{def}} (\{1,2,3,4\},\{(1,2),(1,3),(2,4),(3,4)\})$. Note that vertices have to be renamed. Let $n =_{\mathrm{def}} \dim M$, and let $r$ denote the length of the largest entry in $M$, i.e., the number of bits representing the largest entry of $M$. We construct graph $G$ as follows. For every $x \in \{1,\dots,n\}$, let $G$ have a vertex with name $x$ and a copy of the graph $Z_{r-1}$. Add an arc from $x$ to the source vertex of its $Z_{r-1}$ copy. For every entry $M_{ij}$ of $M$ add, as it is shown in Figure 3, paths to $G$ such that the number of $i,j$-paths of length $2r$ is exactly the number $M_{ij}$. Note that $G$ has at most $n\cdot 3r + n^2\cdot 2r$ vertices, and every vertex has a name of length logarithmic in $\omega$. It is clear that $G$ can be computed using only logarithmic space and that further vertices that are not adjacent to any other vertex can be added to $G$ such that the final graph contains exactly $3+3\log\omega \geq 5rn^2$ vertices (in order to make the names of the vertices a complete interval of $\mathbb{N}$); let $G'$ be the final graph. Then, the number of $i,j$-walks of length $2kr$ in $G'$ is equal to $(M^k)_{ij}$. The product $2kr$ can be computed in logarithmic space, since multiplication is an FL-function. Then, $(M,k,a,i,j)$ in $\mathrm{POM}(\alpha)$ if and only if $(G',2kr,a,i,j)$ in $\mathrm{NoW}(\alpha)$.

■

## 4.2 The complexity of number-of-walks and power-of-matrix problems

For the complexity of our problems, we distinguish between $\alpha=1$ and $\alpha=2$. To be more precise, in case of $\alpha=1$, we will determine the complexity of the problem $\mathrm{NoW}(1)$, and from this complexity, we will obtain all further desired results. In case of $\alpha=2$, we will instead consider the matrix problems and determine the complexities of $\mathrm{POM}(2)$ and $\mathrm{xPOM}(2)$. We start with the case $\alpha=1$. A central idea is to use an interesting result by Fredman, Komlós, Szemerédi, which is stated in Lemma 4.7.

**Lemma 4.7 (Fredman, Komlós and Szemerédi, [29])**
*Let $S \subseteq \mathbb{N}\setminus\{0\}$ be non-empty and finite. Let $n =_{\mathrm{def}} |S|$, and let $m =_{\mathrm{def}} \max S$. Then, there is a prime number $p < n^2\log m$ such that no two elements of $S$ lie in the same residue class modulo $p$.*

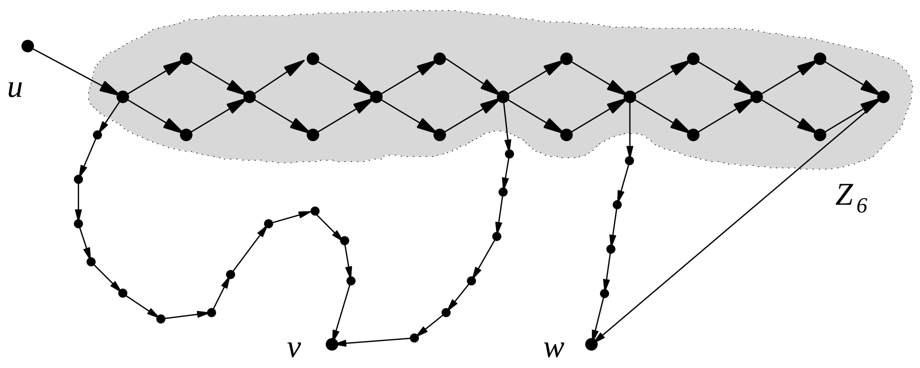


**Figure 3** A graph with nine $u, v$-walks and 80 $u, w$-walks, and all these walks have the same length.

**Proof:** Let $S = \{x_1, \dots, x_n\}$. Let

$$T =_{\rm def} \Big(\prod_{i=1}^{n} x_i\Big) \cdot \Big(\prod_{1 \le i < j \le n} (x_i - x_j)\Big).$$

It is clear that

$$\log |T| \le n \log m + \binom{n}{2} \log m = \binom{n+1}{2} \log m \le n^2 \log m .$$

For a number $x$, let $Q(x)$ denote the set of prime numbers smaller than $x$. Then, by some number-theoretic result, it holds that

$$\log\Big(\prod_{q \in Q(x)} q\Big) = x + o(x),$$

i.e.,

$$n^2 \log m < \log\Big(\prod_{q \in Q(n^2 \log m)} q\Big).$$

Hence, there is some prime number $p$ in $Q(n^2 \log m)$ that does not divide $T$. It follows that no number from $S$ lies in the residue class 0 modulo $p$ and no two numbers from $S$ lie in the same residue class modulo $p$, which proves the lemma.

■

Allender, Reinhardt, Zhou considered a restricted variant of problem NoW(1). They were interested in the number of paths in acyclic graphs. They used Lemma 4.7 to obtain an NL-algorithm [2]. In our case, we have to resolve the problem how to find sufficiently many walks of a certain length, if this length is given in binary representation. The following result shows that we only have to find enough walks of a certain *short* length.

**Lemma 4.8** *Let $G = (V, A)$ be a directed graph, and let $n =_{\rm def} |V|$. Let $k, r \in \mathbb{N}$, $2n^2 \le r < k - n$, and let $W_1 = (u_0, \dots, u_k)$ and $W_2 = (v_0, \dots, v_k)$ be two walks in $G$ such that $u_i = v_i$ for all $i \in \{0, \dots, r\} \cup \{k\}$ and $u_{r+1} \ne v_{r+1}$. Then, there are $r' \in \{r - 2n^2, \dots, r-1\}$ and a walk $W = (w_0, \dots, w_k)$ in $G$ such that $u_i = w_i$ for all $i \in \{0, \dots, r'\} \cup \{k\}$ and $u_{r'+1} \ne w_{r'+1}$.*

**Proof:** Let $s_1, s_2 \in \{r - 2n^2 + 0, \dots, r - 2n^2 + n\}$, $s_1 < s_2$, such that $u_{s_1} = u_{s_2}$. Let $d =_{\rm def} s_2 - s_1$. If there is a $u_{s_1}, u_{s_2}$-walk of length $d$ in $G$ that is different from $W_u =_{\rm def} (u_{s_1}, u_{s_1+1}, \dots, u_{s_2})$, replace this sequence in $W_1$ by the other walk and obtain $W$. Otherwise, $W_u$ is the unique $u_{s_1}, u_{s_2}$-walk of length $d$ in $G$.

Consider the sequence $u_{s_2}, u_{s_2+d}, u_{s_2+2d}, \dots, u_{s_2+ed}$ where $r - d < s_2 + ed \le r$. Note that $e \ge n$. Hence there are $a_1, a_2 \in \{0, \dots, e\}$, $a_1 < a_2$, such that $u_{s_2+a_1d} = u_{s_2+a_2d}$. If $a_1$ and $a_2$ can be chosen such that there is $a \in \{a_1, \dots, a_2\}$ such that $u_{s_2+ad} \ne u_{s_2}$, then extract $(u_{s_2+a_1d}, u_{s_2+a_1d+1}, \dots, u_{s_2+a_2d})$ from $W_1$, insert $a_2 - a_1$

copies of $W_u$ after $u_{s_2}$ and obtain $W$. Otherwise, $u_{s_1} = u_{s_2} = u_{s_2+d} = \cdots = u_{s_2+(e-n+1)d}$.

Let $c_1, c_2 \in \{k-n, \ldots, k\}$, $c_1 < c_2$, such that $u_{c_1} = u_{c_2}$, and let $d' =_{\mathrm{def}} c_2 - c_1$. Note that $d' \leq e - n + 1$. Obtain $W'$ from $W_1$ by deleting the subwalk $(u_{s_1+1}, u_{s_1+2}, \ldots, u_{s_1+d'd})$ and adding $d$ copies of $(u_{c_1+1}, u_{c_1+2}, \ldots, u_{c_2})$ at position $c_1$ in $W_1$. Certainly, $W'$ is a $u_0, u_k$-walk of length $k$. If $W'$ is different from $W_1$, we found walk $W$. Otherwise, $u_{s_1} = u_{s_2} = u_{s_2+d} = u_{s_2+2d} = \cdots = u_{s_2+(e+1)d}$ by induction.

Now, $W_1$ and $W_2$ differ in $u_{r+1}$ and $v_{r+1}$. So, we repeat the construction of the last case with $W_2$ instead of $W_1$ and obtain the desired walk $W$, since $W_u$ is unique.

■

For a directed graph $G = (V, A)$ and two vertices $u, v$ of $G$, let $\#_G(\overrightarrow{uv}, k)$ denote the number of $u, v$-walks of length $k \geq 0$ in $G$.

**Corollary 4.9** *Let $G = (V, A)$ be a directed graph, and let $n =_{\mathrm{def}} |V|$. Let $u, v \in V$ be two vertices of $G$, and let $\nu$ and $k$ be such that $0 \leq 2\nu n^2 + n \leq k$. If $\nu \leq \#_G(\overrightarrow{uv}, k)$, then*

$$\nu \leq \sum_{x,z \in V} \#_G(\overrightarrow{ux}, 2\nu n^2) \cdot W_G(\overrightarrow{xz}, k - 2\nu n^2 - n) \cdot \#_G(\overrightarrow{zv}, n)\,.$$

**Proof:** Observe the following equality. It holds that

$$\nu = \sum_{x,z \in V} \#_G(\overrightarrow{ux}, 2\nu n^2) \cdot \#_G(\overrightarrow{xz}, k - 2\nu n^2 - n) \cdot \#_G(\overrightarrow{zv}, n)\,.$$

For proving the claimed inequality, we distinguish two cases. Let $x, z \in V$ be two vertices of $G$ such that

$$W_G(\overrightarrow{ux}, 2\nu n^2) \cdot \#_G(\overrightarrow{xz}, k - 2\nu n^2 - n) \cdot W_G(\overrightarrow{zv}, n) \geq 2\,.$$

Then, there are two $u, v$-walks of length $k$ in $G$ that coincide on the first $2\nu n^2 + 1$ vertices and differ in a position that is not among the last $n + 1$ ones. Hence, Lemma 4.8 can be applied, which shows the existence of a $u, v$-walk of length $k$ in $G$ that differs from the two first walks not before position $2\nu n^2 - 2n^2$. Repeated application of Lemma 4.8 shows the existence of $\nu$ $u, v$-walks that pairwise differ somewhere at the first $2\nu n^2 + 1$ positions. These walks are partitioned into up to $n$ sets that are determined by the vertex at position $2\nu n^2 + 1$. Then, the claim follows immediately.

Now, assume that, for every pair $x, z$ of vertices of $G$,

$$W_G(\overrightarrow{ux}, 2\nu n^2) \cdot \#_G(\overrightarrow{xz}, k - 2\nu n^2 - n) \cdot W_G(\overrightarrow{zv}, n) \leq 1\,.$$

Then, for every pair $x, z$ of vertices of $G$, if there is an $x, z$-walk of length $k - 2\nu n^2 - n$ in $G$, then it is unique, or one of the other required walks does not exist. Hence,

$$W_G(\overrightarrow{xz}, k - 2\nu n^2 - n) = \#_G(\overrightarrow{xz}, k - 2\nu n^2 - n)$$

for every pair $x, z$ of vertices for which $W_G(\overrightarrow{ux}, 2\nu n^2) = W_G(\overrightarrow{zv}, n) = 1$, and the inequality in fact is an equality.

■

To solve the problem NoW(1) we will apply the method of Allender, Reinhardt, Zhou [2]. Let $G$ be a directed graph. Our chosen representation of $G$ establishes an ordering on the vertices of $G$ by their names. So, every walk in $G$ can uniquely be interpreted as a natural number greater than 0 in $(n{+}1)$-ary representation, where $n$ is the number of vertices of $G$. Hence, we can determine the residue class modulo some number for every walk. Let ResidueOfWalk be the set of tuples $(G, k, q, r, u, v)$ where $G = (V, A)$ is a directed graph, $k, q, r \in \mathbb{N}$, $k$ and $q$ are given in unary representation and $u$ and $v$ are vertices of $G$ and there is a $u, v$-walk of length $k$ in $G$ whose $(n{+}1)$-ary number representation for $n$ the number of vertices of $G$ lies in the residue class $r$ modulo $q$.

**Lemma 4.10** ResidueOfWalk *is in* NL .

**Proof:** Let $(G, k, q, r, u, v)$ be an instance of ResidueOfWalk. If $r \geq q$ compute $r' \in \{0, \ldots, q-1\}$ such that $r' \equiv r \pmod{q}$ and replace $r$ by $r'$. Guess a $u, v$-walk of length $k$ iteratively, i.e., vertex by vertex, and compute its residue class modulo $q$ using the standard division algorithm. Since $k$ and $q$ are given in unary representation, this algorithm needs only logarithmic space.

■

**Theorem 4.11** NoW(1) *is in* NL .

**Proof:** Let $(G, k, \nu, u, v)$ be an instance of NoW(1). Let $n =_{\text{def}} |V|$. By Lemma 4.3, Walk is in NL. To decide whether $\#_G(\overrightarrow{uv}, k) \geq \nu$ it suffices, due to Corollary 4.9, to guess numbers $\nu_w \leq \nu$ and $\nu'_w \leq \nu$ for all vertices $w$ of $G$ such that $\nu_w \leq \#_G(\overrightarrow{uw}, 2\nu n^2)$ and $\nu'_w \leq \#_G(\overrightarrow{wv}, n)$ and

$$\nu \leq \sum_{x,z \in V} \nu_x \cdot W_G(\overrightarrow{xz}, k - 2\nu n^2 - n) \cdot \nu'_z \,.$$

Remember that $\nu$ is given in unary representation, so that the guessed numbers can be represented in binary form in logarithmic space. To verify the sufficiency of each guessed $\nu_w$ for which $W_G(\overrightarrow{wv}, k - 2\nu n^2 - n) = 1$ holds we apply the following logarithmic-space algorithm with oracle ResidueOfWalk. Note that all walks of length $2\nu n^2$ have a numerical representation with value not exceeding $(n+1)^{2\nu n^2} = 2^{2\nu n^2 \cdot \log(n+1)} \leq 2^{2\nu n^3}$. Applying Lemma 4.7, we have to find a prime number $p$ smaller than $(\nu_w)^2 \cdot 2\nu n^3$ such that there are $\nu_w$ residue classes modulo $p$ each containing the numerical representation of a $u, w$-walk of length $2\nu n^2$. This can be done in logarithmic space. The numbers $\nu'_w$ are verified similarly. Since $\mathrm{L}^{\mathrm{NL}} = \mathrm{NL}$ by Theorem 3.13, NoW(1) is in NL.

■

**Corollary 4.12** NoW(1), *x*NoW(1), NoMW(1) *and* xNoMW(1) *and* POM(1) *and* xPOM(1) *are* NL*-complete.*

**Proof:** For hardness, we first show that NoW(1) and xNoW(1) are NL-hard. Let $(G, u, v)$ be an instance of GAP and GIP. It holds that $G$ contains a $u, v$-path if and only if there is $k < n$ for $n$ the number of vertices of $G$ such that $G$ contains a $u, v$-path of length $k$. Let $G'$ emerge from $G$ by adding the loop $(u, u)$. Hence, $G$ has a $u, v$-path if and only if $G'$ has a $u, v$-walk of length $n$, i.e., $(G, u, v)$ in GAP if and only if $(G', n, 1, u, v)$ in NoW(1). In case of xNoW(1), it holds that $(G, u, v)$ in GIP if and only if $(G', n, 0, u, v)$ in xNoW(1). Applying Lemmata 4.2 and 4.6, we conclude NL-hardness for all problems.

Due to Theorem 4.11, NoW(1) is in NL. Applying Lemma 4.1, xNoW(1) is in NL. By Lemma 4.6, POM(1) and xPOM(1) are in NL. Finally, since NL is closed under $\leq_m^{\mathrm{NL}}$-reducibility due to Lemma 3.14, NoMW(1) and xNoMW(1) are in NL due to Theorem 4.4.

■

Our solution of case $\alpha = 2$, i.e., determining the complexity of problems like NoW(2), is less exact than the case considered before. We will not give a complete solution, since we will only be able to give upper and lower complexity bounds for NoW(2) that do not meet each other. A complete solution remains one of the major open problems of this thesis.

**Lemma 4.13** POM(2) *is in* P.

**Proof:** Let $(M, k, a, i, j)$ be an instance of POM(2). Let $n$ be the dimension of $M$. We have to answer the question whether $(M^k)_{ij} \geq a$ does hold. Let $M'$ emerge from $M$ by replacing every entry that is larger than $a$ by $a+1$. We denote this operation by $[M]_{\leq a}$, i.e., $[M]_{\leq a} = M'$. Then, $(M^k)_{ij} \geq a$ if and only if $((M')^k)_{ij} \geq a$. It holds that

$$[M^r]_{\leq a} = [[M^i]_{\leq a} \cdot [M^{r-i}]_{\leq a}]_{\leq a}$$

for every $r \geq 0$ and $i \in \{0, \ldots, r\}$. Observe that the left hand side matrix can be represented using only $b \cdot n^2$ bits where $b$ is the number of bits for representing $a+1$ in binary form. Furthermore, for $r \geq 0$, $M^{2r} = M^r \cdot M^r$ and $M^{2r+1} = M^r \cdot M^r \cdot M$. Multiplying two matrices takes polynomial time, and computing $[\cdot]_{\leq a}$ takes polynomial time. Since $[M^k]_{\leq a}$ can be computed using at most $2c$ matrix multiplications where $c$ is the number of (significant) digits of the binary representation of $k$, $[M^k]_{\leq a}$ can be computed in polynomial time. Hence, the size of the value of $(M^k)_{ij}$ with respect to $a$ can be determined in polynomial time.

■

The main observation that we have to make in connection with the algorithm of Lemma 4.13 is that it is not possible to directly compute $M^k$ for a matrix $M$ and a number $k$, since entries of $M^k$ can become very large for large $k$. Furthermore, it is easy to see that problem POM(2) can be modified such that the queried entry of the matrix power is output, if it does not exceed number $a$. So, we obtain a function problem corresponding to xPOM(2). Alternatively, using POM(2) and binary search, we can also compute that value in polynomial time, if it does not exceed $a$.

**Theorem 4.14** NoW(2), NoMW(2) *and* POM(2) *are in* P *and* PL-*hard.*

**Proof:** Similar to the proof of Corollary 4.12, the problems NoW(2), NoMW(2) and POM(2) are in P due to Lemma 4.13. For hardness, it suffices to show that NoW(2) is PL-hard, since hardness translates to the other problems due to Lemmata 4.2 and 4.6. Let $A \in \mathrm{PL}$. Then, there are functions $f \in \#\mathrm{L}$ and $g \in \mathrm{FL}$ such that $x \in A$ if and only if $f(x) \geq g(x)$. Let $f$ be computed by nondeterministic logarithmic-space Turing machine $M$. Applying Lemma 3.7, in logarithmic space, we obtain an acyclic graph $G_{M,x}$ and vertices $u$ and $v$ of $G_{M,x}$ such that the number of $u, v$-paths in $G_{M,x}$ is equal to $f(x)$. Adding a loop to vertex $v$, we obtain $G'_{M,x}$, and there is a constant $c$ independent of $x$ such that $f(x)$ is equal to the number of $u, v$-walks of length $|x|^c$ in $G'_{M,x}$. Hence, $x \in A$ if and only if $(G'_{M,x}, |x|^c, g(x), u, v) \in \mathrm{NoW}(2)$, which defines a logarithmic-space reduction.

■

**Corollary 4.15** xNoW(2), xNoMW(2) *and* xPOM(2) *are in* P *and* $\mathrm{C_{=}L}$-*hard.*

**Proof:** The proof for containment and hardness is analogous to the proof of Theorem 4.14. It only has to be taken into account that $A \in \mathrm{C_{=}L}$ if and only if there are functions $f \in \#\mathrm{L}$ and $g \in \mathrm{FL}$ such that $x \in A$ if and only if $f(x) = g(x)$.

■

## 4.3 Two NP-complete problems

There is a huge number of NP-complete problems, and only a very few of them can be found in the classical book by Garey and Johnson [31]. The most popular among these problems is SAT, the problem that asks for satisfiability of Boolean formulas given in conjunctive normal form. Without loss of generality, we assume that instances of SAT do not contain clauses with two appearances of the same variable. An interesting and very useful restriction of instances of SAT is denoted by 3-SAT: 3-SAT contains all elements from SAT that have exactly three literals per clause.

**Theorem 4.16 (Cook, [17])**
SAT *and* 3-SAT *are* NP-*complete.*

We define two new problems that turn out to be NP-complete, too. The first one can be considered a decisional variant and generalisation of the Chinese Remainder

Theorem. The Chinese Remainder Theorem guarantees the existence of a solution for a system of congruence equations where the involved moduli are pairwise relatively prime numbers. Two natural numbers are *relatively prime* if their greatest common divisor is 1.

**Theorem 4.17 (Chinese Remainder Theorem)**
*Let $b_1, \dots, b_k$ be pairwise relatively prime numbers, and let $n_1, n_2 \in \mathbb{N}$. Let $b =_{\mathrm{def}} \prod_i b_i$. Then, $n_1 \equiv n_2 \pmod{b}$ if and only if $n_1 \equiv n_2 \pmod{b_i}$ for every $i \in \{1, \dots, k\}$.*

For our problem, that will be called Set-SCE, we allow arbitrary moduli and a set of remainders and ask for satisfaction.

Problem definition

Set-SCE (satisfiability of a set-system of congruence equations).
**Instance** $((A_1, b_1), \dots, (A_k, b_k))$ where $A_1, \dots, A_k$ are finite sets of natural numbers, and $b_1, \dots, b_k$ are natural numbers greater than 1 given in unary representation.
**Question** Are there $n \in \mathbb{N}$ and $a_1 \in A_1, \dots, a_k \in A_k$ such that $n \equiv a_i \pmod{b_i}$ for all $i \in \{1, \dots, k\}$?

Note that it is not important to require binary representation of the numbers in $A_1, \dots, A_k$. However, we assume a binary representation of them, only to fix a system. For numbers $b, c \in \mathbb{N}$ and set $A \subseteq \mathbb{N}$, we say that $c \equiv A \pmod{b}$ holds if there is $a \in A$ such that $c \equiv a \pmod{b}$. Hence, the question of Set-SCE can be reformulated as: Is there $n \in \mathbb{N}$ such that $n \equiv A_i \pmod{b_i}$ for all $i \in \{1, \dots, k\}$?

**Lemma 4.18** 3-Sat $\leq_m^{\mathrm{L}}$ Set-SCE .

**Proof:** Let $H = H(x_1, \dots, x_r)$ be an instance of 3-Sat, i.e., a Boolean formula in conjunctive normal form with exactly three literals per clause, and no variable appears twice in a clause. The variables in $H$ are $x_1, \dots, x_r$.

[Construction of an instance of Set-SCE.]
Let $k$ be the number of clauses in $H$, and let $K_1, \dots, K_k$ be the clauses in $H$. Let $p_1, \dots, p_r$ be $r$ prime numbers. By indices, there is a 1-to-1 correspondence between the variables of $H$ and the prime numbers. For every $j \in \{1, \dots, k\}$, let:

$$b_j =_{\mathrm{def}} \prod_{i=1}^{r} \begin{cases} p_i & \text{, if } x_i \text{ is variable in } K_j \\ 1 & \text{, if } x_i \text{ is not variable in } K_j \,. \end{cases}$$

Variables in each clause are ordered according to their indices; this means that there is a first, a second, a third variable in each clause. Let $f_j = f_j(z_1, z_2, z_3)$ be the Boolean function defined by $K_j$, where $z_1$ corresponds to the first, $z_2$ corresponds to the second and $z_3$ corresponds to the third variable in $K_j$. Let $x_j^{(1)}, x_j^{(2)}, x_j^{(3)}$ be the variables in $K_j$, and let $p_j^{(1)}, p_j^{(2)}, p_j^{(3)}$ be the corresponding primes. Let $N_j$ be the set of natural numbers $a$ smaller than $b_j$ such that $a \equiv \{0, 1\} \pmod{p_j^{(i)}}$ for all

$i \in \{1,2,3\}$. Note that, by the Chinese Remainder Theorem, $N_j$ contains exactly eight numbers, among them 0 and 1. We define $A_j$ as the set of numbers $a \in N_j$ such that $f_j(c_{j,a}^{(1)}, c_{j,a}^{(2)}, c_{j,a}^{(3)}) = 1$ where

$$c_{j,a}^{(i)} =_{\text{def}} \begin{cases} 1 & \text{, if } a \equiv 0 \pmod{p_j^{(i)}} \\ 0 & \text{, if } a \equiv 1 \pmod{p_j^{(i)}} \end{cases}$$

for $i \in \{1,2,3\}$. Then, $\mathcal{S}_H =_{\text{def}} ((A_1, b_1), \ldots, (A_k, b_k))$ is an instance of SET-SCE.

[Satisfiability of $H$ translates to $\mathcal{S}_H$.]

Let $\beta$ be a satisfying assignment for $H$. Due to the Chinese Remainder Theorem there is a number $n < p_1 \cdots p_r$ that satisfies the congruence equations

$$n \equiv 1 - \beta(x_i) \pmod{p_i}$$

for all $i \in \{1, \ldots, r\}$. Let $j \in \{1, \ldots, k\}$, and let $p_j^{(1)}, p_j^{(2)}, p_j^{(3)}$ correspond to the variables $x_j^{(1)}, x_j^{(2)}, x_j^{(3)}$ in $K_j$. Let $a_j < b_j$ satisfy

$$a_j \equiv 1 - \beta(x_j^{(i)}) \pmod{p_j^{(i)}}$$

for every $i \in \{1,2,3\}$. Observe that $a_j \in A_j$. Since the remainders of $a_j$ and $n$ correspond on $p_j^{(1)}$, $p_j^{(2)}$ and $p_j^{(3)}$, $n \equiv a_j \pmod{b_j}$ due to the Chinese Remainder Theorem. Hence, $n$ is a solution of $\mathcal{S}_H$.

[Satisfiability of $\mathcal{S}_H$ translates to $H$.]

Let $n$ be a solution of $\mathcal{S}_H$. We show that $\beta$, defined as

$$\beta(x_i) =_{\text{def}} \begin{cases} 1 & \text{, if } n \equiv 0 \pmod{p_i} \\ 0 & \text{, if } n \not\equiv 0 \pmod{p_i} \end{cases}$$

for all $i \in \{1, \ldots, r\}$, satisfies $H$. Let $a_j \in A_j$ such that $n \equiv a_j \pmod{b_j}$, and let $x_i$ be a variable in $K_j$ such that $\Lambda \in \{x_i, \neg x_i\}$ is literal in $K_j$ and $\Lambda = x_i$ if and only if $a_j \equiv 0 \pmod{p_i}$. Then, $\Lambda = x_i$ if and only if $n \equiv 0 \pmod{p_i}$ by the Chinese Remainder Theorem. Therefore, $K_j$ is satisfied by $\beta$.

[Logarithmic-space computable reduction.]

It remains to show that the reduction, from $H$ to $\mathcal{S}_H$, can be carried out in logarithmic space. The number set $\{2, \ldots, r^2\}$ contains $r$ primes (if $r \geq 2$), so that in logarithmic space $r$ prime numbers can be found. The sets $A_1, \ldots, A_k$ and the numbers $b_1, \ldots, b_k$ can be constructed by straightforward enumeration and verification in logarithmic space.

■

Our second problem is closely related to SET-SCE. This problem is called SORSR and asks a question about a given system of registers. (These registers are different from registers of random access machines.) A *register* $R$ of size $k \geq 1$ is a vector with $k$ components. The entries of $R$ are denoted as $R(1), \ldots, R(k)$. A *left* *ring-shift* operation is performed on a register by moving the contents of every but the first components one position to the left and the previously first entry is moved to the rightmost (*last*) position. We assume that the considered register is spread from left to right, i.e., from $R(1)$ on the left side to $R(k)$ on the right side.

Problem definition

soRSR (simultaneous-ones ring-shift registers problem).
**Instance** A set $\{R_1, \dots, R_k\}$ of registers that contain natural numbers.
**Question** Is there a number $r \geq 0$ such that, after performing $r$ left ring-shift operations on each register, all registers have entry 1 in the leftmost component?

Similarly to the problem Set-SCE, it is not necessary to fix a number representation system for soRSR, but for simplicity, we assume binary representation.

**Lemma 4.19** Set-SCE $\leq_m^{\mathrm{L}}$ soRSR.

**Proof:** Let $((A_1, b_1), \dots, (A_k, b_k))$ be an instance of Set-SCE. Note that, since $b_i$ is given in unary form, for every binary number $a$ it can be determined a number $a' \in \{0, \dots, b_i{-}1\}$ such that $a \equiv a' \pmod{b_i}$ using only logarithmic space. Hence, in logarithmic space, sets $A'_1, \dots, A'_k$ can be determined such that $A'_i \subseteq \{0, \dots, b_i{-}1\}$, $i \in \{1, \dots, k\}$, and $a' \in A'_i$ if and only if there is $a \in A_i$ such that $a \equiv a' \pmod{b_i}$. We define $k$ registers $R_1, \dots, R_k$ where $R_i$ is of size $b_i$ for all $i \in \{1, \dots, k\}$. The registers contain 0 and 1 as entries, and $R_i(j) = 1$, $i \in \{1, \dots, k\}$, $j \in \{1, \dots, b_i\}$, if and only if $j{-}1 \in A'_i$. Observe that $R_1, \dots, R_k$ can be computed in logarithmic space.

By definition of the left ring-shift operation and the registers, it holds that $R_i(1) = 1$ after application of $t \geq 0$ left ring-shift operations if and only if there is $j \in \{1, \dots, b_i\}$ such that $R_i(j) = 1$ and $j{-}1 \equiv t \pmod{b_i}$. Then, there is a number $r$ such that, after application of $r$ left ring-shift operations on all registers, all registers contain 1 in their leftmost components if and only if there are $a_1, \dots, a_k$ and $t$ such that $a_i \in \{1, \dots, b_i\}$, $R_i(a_i) = 1$ and $a_i{-}1 \equiv t \pmod{b_i}$ for every $i \in \{1, \dots, k\}$. Hence, $\{R_1, \dots, R_k\} \in$ soRSR if and only if $((A_1, b_1), \dots, (A_k, b_k)) \in$ Set-SCE. ∎

**Theorem 4.20** Set-SCE *and* soRSR *are* NP*-complete.*

**Proof:** NP-hardness of both problems follows from Lemmata 4.18 and 4.19 and Theorem 4.16. So, it suffices to show that soRSR can be accepted in nondeterministic polynomial time.

Let $\{R_1, \dots, R_k\}$ be an instance of soRSR, and let $b_1, \dots, b_k$ be the sizes of the registers $R_1, \dots, R_k$, respectively. Observe that, if there is $t \geq 0$ such that after application of $t$ left ring-shift operations to the registers such that each register has 1 in its leftmost component, there is such a number $t$ smaller than $b_1 \cdots b_k$. So, it suffices to check whether there is $t < b_1 \cdots b_k$ such that we obtain all leftmost components filled with 1 after applying $t$ left ring-shift operations to the registers. If this can be checked in polynomial time, we obtain an overall nondeterministic polynomial-time algorithm.

We can restrict to the question, given a register $R$ of size $b$ and a number $r$, whether $R(1)$ contains 1 after application of $r$ left ring-shift operations. In polynomial time, number $a < b$ such that $r \equiv a \pmod{b}$ can be computed, and accept if and only if $R(a+1) = 1$. This concludes the proof.

■

# Chapter 5
# Introducing finite recurrent systems

Recurrences or recurrent equations appear in many fields of theoretical computer science, mathematics, physics—in fields where discrete processes are modelled and studied. The first and widely known recurrence defines the set of Fibonacci numbers: this sequence starts with twice number 1, and every further element is the sum of its two predecessors, i.e., we obtain 1, 1, 2, 3, 5, 8, 13, and so on. Even though the first few numbers do not seem sensational, the Fibonacci numbers turn out very useful in many areas of science (see, for instance, *The Fibonacci quarterly*, that is a scientific journal devoted to the Fibonacci numbers and related questions).

In the case of Fibonacci numbers, only one recurrent equation is involved. In other cases, one may want to study number sequences whose elements are generated by taking elements from other number sequences that are generated by applying different generation rules. So, the transition from single recurrence equations to recurrent systems, systems of possibly nested recurrence equations, is natural. A formal definition is presented in the second section of this chapter. We will give a rather general definition of recurrent systems. For our investigations, we will restrict to special ones, that fit well into a series of previous work.

We will not consider recurrent systems as the main object of our investigations, but we will use them for concisely defining sets of numbers, about which we wish to obtain information. The corresponding problems are so-called membership problems, and we will define two versions. We will ask whether a given recurrent system generates a given number in some iteration step or whether there is an iteration step in which the given number is generated. Precise definitions and first results are given in Section 5.3. To give an idea, the questions may be whether a given number is the $k$-th Fibonacci number or whether it *is* a Fibonacci number.

Complexity-theoretic results always depend on input representation. Since recurrent systems consist of equations, or functions (depending on the point of view), we choose appropriate arithmetical circuits as representation models. These circuits are defined in the following section. The chapter ends with a summary of complexity results of problems that are closely related to our membership problems.

## 5.1 Arithmetical circuits

Complexity theory knows different mathematical objects for proving upper and lower complexity bounds. A rich and fertile field is created by the study of circuits. Especially, the complexity of basic arithmetical operations, like addition, multiplication and division, can be captured adequately. Also, circuit complexity is able to make finer distinctions compared to classical complexity measures based on Turing machines, for instance. A good introduction to and survey on circuit complexity is provided by Vollmer's book on that subject [83].

Here, however, we will not study arithmetical circuits themselves, nor will we seek for results about them. Instead, we will use them only for representation purposes. Informally spoken, arithmetical circuits are acyclic graphs whose vertices are labelled. We first define arithmetical circuits in a very general style.

**Definition 14** *Let $\mathfrak{M}$ be a set (the* universe*), and let $\mathfrak{F}$ be a set of operations over $\mathfrak{M}$. Let $n \geq 1$. An **$n$-ary arithmetical $\mathfrak{F}$-circuit** over $\mathfrak{M}$ is a tuple $(G, m, g, \alpha)$ where $G = (V, A)$ is an acyclic graph having at least $n$ source vertices, $m$ is a mapping that labels the arcs of $G$ with numbers such that no pair of arcs is assigned the same number, $g$ denotes a vertex of $G$, that is called the* output vertex*, and $\alpha$ is a mapping that labels vertices of $G$ with elements from $\mathfrak{F} \cup \{1, \dots, n\}$ in the following way:*

*(a) the numbers from $\{1, \dots, n\}$ are assigned to $n$ different vertices*

*(b) for every vertex $x$ of $G$:*

*(b1) if $\alpha(x) \in \{1, \dots, n\}$ then $x$ is a source vertex, also called an* input vertex

*(b2) if $\alpha(x) \in \mathfrak{F}$ and $x$ has $k$ in-coming arcs then $\alpha(x)$ is a $k$-ary operation.*

Instead of always speaking of $n$-ary arithmetical $\mathfrak{F}$-circuits, we also speak of *arithmetical circuits*, for short, if $n$ and $\mathfrak{F}$ are not precised. Arithmetical circuits represent functions, in particular, arithmetical $\mathfrak{F}$-circuits represent functions that are built only by using operations from $\mathfrak{F}$. The evaluation of a function represented by an arithmetical circuit on a given input is defined as follows. Let $C = (G, m, g, \alpha)$ be an $n$-ary arithmetical $\mathfrak{F}$-circuit over some set $\mathfrak{M}$. Let $(a_1, \dots, a_n)$ be the input, where $a_1, \dots, a_n$ are elements from $\mathfrak{M}$. The evaluation function $\beta$ assigns an element from $M$ to every vertex $x$ of $G$ in the following way (clearly, it is assumed that $\mathfrak{M}$ is closed under all operations in $\mathfrak{F}$):

- if $\alpha(x) \in \{1, \dots, n\}$, then $\beta(x) =_{\text{def}} a_i$ where $i = \alpha(x)$
- if $\alpha(x) \in \mathfrak{F}$ and $x$ has $k$ in-coming arcs and $e_1, \dots, e_k$ are these arcs, where we assume $m(e_1) < \cdots < m(e_k)$, and $x_1, \dots, x_k$ are the predecessors of $x$ corresponding to $e_1, \dots, e_k$, respectively, then $\beta(x) =_{\text{def}} [\alpha(x)](\beta(x_1), \dots, \beta(x_k))$.

If the vertices are processed according to a topological ordering for $G$, it is clear that the assignment $\beta$ is well-defined. Then, the result of the function represented by circuit $C$ applied to $(a_1, \dots, a_n)$ is $\beta(g)$, i.e., the value that is assigned to the output vertex of $C$. Graph $G$ of $C$ is also called the graph *underlying* $C$. It must be remarked that the graph underlying an arithmetical circuit does not have to be connected. Also, there does not always have to be a path from every input vertex of $C$ to the output vertex. Furthermore, $G$ may contain source vertices that are not labelled input vertices, if $\mathfrak{F}$ contains 0-ary operations.

**Lemma 5.1** *Let $n$ be a number, and let $\mathfrak{F}$ be a set of operations. The problem, given a tuple $C = (G, m, g, \alpha)$, deciding whether $C$ is an $n$-ary arithmetical $\mathfrak{F}$-circuit is in* NL .

**Proof:** In logarithmic space, it can be verified whether the input is of the correct form, i.e., whether it represents a tuple whose components are a directed graph, a mapping from arcs to numbers, a vertex and another mapping from vertices to $\{1,\ldots,n\} \cup \mathfrak{F}$, where the elements of $\mathfrak{F}$ are represented appropriately. If it is the case, we can assume that $(G, m, g, \alpha)$ is the input. Also in logarithmic space, it can be verified whether $m$ assigns every number to at most one arc of $G$, whether $g$ is a vertex of $G$ and whether $\alpha$ maps vertices to $\{1,\ldots,n\} \cup \mathfrak{F}$ in the correct way. Hence, it remains to check whether $G$ is acyclic. This can be done in nondeterministic logarithmic space due to Theorem 3.12. Hence, we obtain an overall nondeterministic logarithmic space algorithm.

■

As already mentioned, we will use arithmetical circuits only for representing functions. As a basic property of the operation set that will be involved it holds that every operation is commutative, i.e., the order of the input values is not significant. So, we will consider arithmetical circuits only as triples, where we suppress the second component, mapping $m$, of our original definition.

An important operation for arithmetical circuits is superposition. More precisely, given two functions represented by arithmetical circuits, how do we obtain an arithmetical circuit representing the superposition of these functions? This operation is rather simple, since it suffices to compute the disjoint union of the two underlying graphs and glue together the output vertex of the one circuit with the appropriate input vertex of the other circuit. Depending on the actual situation, input vertices may have to be glued together or not. An example is provided in Figure 4. The picture shows four schemes of arithmetical circuits: three 3-ary circuits, $C_1$, $C_2$ and an unnamed one (the rightmost circuit in the picture), and a 2-ary circuit, $C$. Let $f_{C_1}$, $f_{C_2}$ and $f_C$ denote the functions represented by $C_1$, $C_2$ and $C$, respectively. Then, the unnamed circuit represents function $f(x_1, x_2, x_3) = f_C(f_{C_1}(x_1, x_2, x_3), f_{C_2}(x_1, x_2, x_3))$. Note that the input vertices of $C$ have become so-called *inner vertices*, i.e., vertices that are not input vertices, by a glueing operation.

## 5.2 Finite recurrent systems over sets of natural numbers

A recurrence is a pair composed of a function and a set of initial values. Using recurrences, one can generate infinite sequences of objects by applying the function to certain of already generated objects. Usual recurrences are defined over natural, real or complex numbers and involve only basic arithmetical operations like addition and multiplication. We extend this notion to recurrent systems over sets of natural numbers. Remember that 0 is also a natural number.

**Definition 15** *Let $n \geq 1$. A **finite recurrent system** over sets of natural numbers of dimension $n$ is a pair $S = (\mathcal{F}, A)$ where $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ for $f_1, \ldots, f_n$ $n$-ary functions over sets of natural numbers and $A \in \mathbb{N}^n$. The dimension $n$ of recurrent*

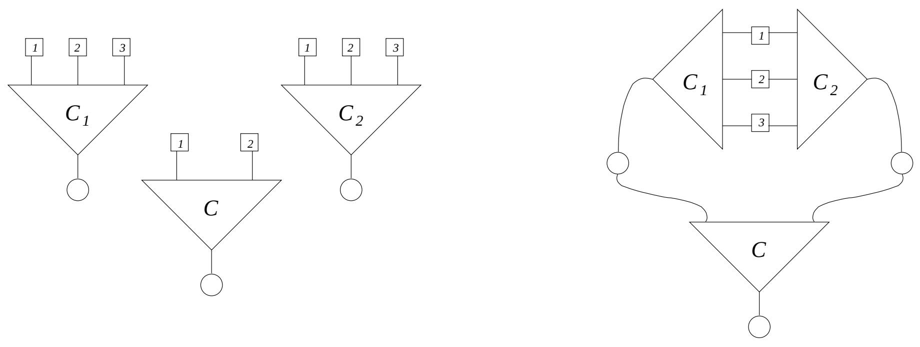


**Figure 4** Circuit representation of the superposition of functions. Input vertices are represented as squares.

*system $S$ is denoted as* $\dim S$.

Finite recurrent systems define sequences of tuples of sets of numbers. These tuples are generated by parallelly applying the functions from $\mathcal{F}$. Formally, this process is defined as follows. Let $S = (\mathcal{F}, A)$ be a finite recurrent system over sets of natural numbers of dimension $n$ where $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ and $A = (a_1, \ldots, a_n)$. We define, for every $t \geq 0$:

$$\begin{aligned} S_i(0) &=_{\text{def}} S[f_i](0) =_{\text{def}} \{a_i\}, \;\; i \in \{1, \ldots, n\} \\ S_i(t+1) &=_{\text{def}} S[f_i](t+1) =_{\text{def}} f_i(S_1(t), \ldots, S_n(t)), \;\; i \in \{1, \ldots, n\} \end{aligned}$$

$$\begin{aligned} \mathcal{F}(t) &=_{\text{def}} (S_1(t), \ldots, S_n(t)) \\ S(t) &=_{\text{def}} S_n(t)\,. \end{aligned}$$

In words, a recurrent system over sets of natural numbers generates tuples containing sets of natural numbers in each component. Starting with the initial values, that are determined by the second component of a finite recurrent system, the functions of a recurrent system are simultaneously applied to the last obtained tuple. Compounding the results of all functions yields a new tuple, that is also the input for the next iteration step. If we define concrete recurrent systems, it is not always best to denote the involved functions as $f_1, \ldots, f_n$. Instead, we may give more convenient names, and the denotation $S[h]$ is applied in these cases.

To give names, we often say that $f_n$ is the *output function of $S$*. Furthermore, we call $S_i(t)$ the *result of the $i$-th function of recurrent system $S$ in the $t$-th iteration step*, and $S(t)$ is the *result of the system in the $t$-th iteration step.* Compounding the results of the system of all iteration steps, i.e., compounding the results of the output function of $S$, we obtain a sequence of sets of natural numbers; this sequence is *represented* or *defined* by $S$. By $[S]$, we denote the union of these sets, i.e., $[S] =_{\text{def}} \bigcup_{t \geq 0} S(t)$. So, a finite recurrent system defines a sequence of sets of numbers as well as a set of numbers. We are interested in two problems that are immediately implied by our definitions. One can ask whether a number $b$ is generated in iteration step $t$ or whether $b$ is generated in some iteration step at all.

Several authors studied so-called membership problems for sets of natural numbers that can be built from singleton sets by applying the set operations union, intersection, complementation and the two arithmetical set operations addition and multiplication, denoted by $\oplus$ and $\otimes$, respectively, [79], [84], [87], [61]. Addition and multiplication on sets are defined elementwise in the following sense. Let $A, B \subseteq \mathbb{N}$. Then,

$$\begin{aligned} A \oplus B &=_{\text{def}} \{r + s : r \in A \text{ and } s \in B\} \\ A \otimes B &=_{\text{def}} \{r \cdot s : r \in A \text{ and } s \in B\}\,. \end{aligned}$$

Let $\mathcal{O} \subseteq \{\cup, \cap, ^-, \oplus, \otimes\}$. An $n$-ary $\mathcal{O}$-function $f = f(x_1, \ldots, x_n)$ is a function over the variables $x_1, \ldots, x_n$ defined by using only operations from $\mathcal{O}$. We will consider finite recurrent systems that are built using only $\{\cup, \cap, ^-, \oplus, \otimes\}$-functions.

**Definition 16** *Let $\mathcal{O} \subseteq \{\cup, \cap, ^-, \oplus, \otimes\}$, and let $n \geq 1$. A* ***finite recurrent $\mathcal{O}$-system*** *$S = (\mathcal{F}, A)$ over sets of natural numbers of dimension $n$ is a finite recurrent system over sets of natural numbers of dimension $n$ where every function in $\mathcal{F}$ is an $\mathcal{O}$-function.*

Since we will mostly deal with finite recurrent $\{\cup, \cap, ^-, \oplus, \otimes\}$-systems over sets of natural numbers we will henceforth call them *recurrent systems*, for short. If we want to restrict the involved functions to special operation sets, we will speak of *recurrent $\mathcal{O}$-systems* instead of "finite recurrent $\mathcal{O}$-systems over sets of natural numbers".

It is high time to discuss an example. The possibly most famous recurrence known in mathematics defines the Fibonacci numbers. The first few values are:

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ \ldots\ ;$$

in general, $F(n+2) =_{\text{def}} F(n+1) + F(n)$, $n \geq 0$, where $F(0) =_{\text{def}} F(1) =_{\text{def}} 1$. Here, the nature of a usual recurrence is well comprehensible. We want to define a recurrent system that computes the same sequence. It is already clear by definition that recurrent systems do not generate numbers but sets of numbers. However, in case of singleton sets a correspondence is easy to establish. Let

$$\begin{aligned}&\mathcal{F} =_{\text{def}} \langle f_1, f_2 \rangle \quad \text{where} \quad f_1(x_1, x_2) =_{\text{def}} x_2 \ \text{ and } \ f_2(x_1, x_2) =_{\text{def}} x_1 \oplus x_2\\ &A =_{\text{def}} (0, 1)\,.\end{aligned}$$

Observe that $f_1$ and $f_2$ are 2-ary $\{\oplus\}$-functions. Using $\mathcal{F}$ and $A$, we define the following recurrent $\{\oplus\}$-system: $S =_{\text{def}} (\mathcal{F}, A)$. Consider the evaluation of the first few iteration steps of $S$:

$$\begin{aligned}
S_1(0) &= \{0\}\\
S(0) = S_2(0) &= \{1\}\\
S_1(1) &= f_1(S_1(0), S_2(0)) = f_1(\{0\}, \{1\}) = \{1\}\\
S(1) = S_2(1) &= f_2(S_1(0), S_2(0)) = f_2(\{0\}, \{1\}) = \{0\} \oplus \{1\} = \{1\}\\
S_1(2) &= f_1(S_1(1), S_2(1)) = f_1(\{1\}, \{1\}) = \{1\}\\
S(2) = S_2(2) &= f_2(S_1(1), S_2(1)) = f_2(\{1\}, \{1\}) = \{1\} \oplus \{1\} = \{2\}\,.
\end{aligned}$$

So, it holds that $S(0) = \{F(0)\}$, $S(1) = \{F(1)\}$, $S(2) = \{F(2)\}$, and by induction, it can indeed be shown that $S(t) = \{F(t)\}$ for every $t \geq 0$.

An operation, that is used sometimes, is the extension of a recurrent system by further components. Formally, this is defined as follows. Let $\mathcal{O} \subseteq \{\cup, \cap, ^-, \oplus, \otimes\}$. Let $S = (\mathcal{F}, A)$ be a recurrent $\mathcal{O}$-system of dimension $n$, where $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ and $A = (a_1, \ldots, a_n)$. Let $\mathbf{x}' =_{\text{def}} (x_1, \ldots, x_n, x_{n+1})$, let $f = f(\mathbf{x}')$ be an $\mathcal{O}$-function, and let $a \in \mathbb{N}$. The result of adding $\langle f, a \rangle$ as $(n+1)$-th component to $S$ is the recurrent $\mathcal{O}$-system $S' = (\mathcal{F}', A')$ where $\mathcal{F}' = \langle f_1', \ldots, f_n', f_{n+1} \rangle$ and $A' = (a_1, \ldots, a_n, a)$ and $f_i'(\mathbf{x}') = f_i(x_1, \ldots, x_n)$ for every $i \in \{1, \ldots, n\}$. It is obvious that adding a component to a recurrent system can be done by a logarithmic-space computable function. A very analogue definition holds if two (or more) components are added to a recurrent system.

## 5.3 Membership problems for finite recurrent systems

For recurrent systems we define two types of membership problems. The *existential membership problem* $\mathrm{M}_{ex}$ for recurrent systems asks, given a recurrent system $S$ and a number $b$, whether $b$ is contained in set $[S]$ defined by recurrent system $S$. The *exact membership problem* $\mathrm{M}_{tm}$ asks, given a recurrent system $S$ and numbers $t$ and $b$, whether $b$ is contained in the result of the $t$-th iteration step of $S$. We want to study the complexities of these membership problems with respect to the involved functions. That is why we define classes of existential and exact membership problems. Let $\mathcal{O} \subseteq \{\cup, \cap, ^{-}, \oplus, \otimes\}$.

$$\mathrm{M}_{ex}(\mathcal{O}) =_{\mathrm{def}} \{(S,b) : \quad S \text{ a recurrent } \mathcal{O}\text{-system and } b \in [S]\}$$
$$\mathrm{M}_{tm}(\mathcal{O}) =_{\mathrm{def}} \{(S,t,b) : S \text{ a recurrent } \mathcal{O}\text{-system and } b \in S(t)\}$$

Instead of writing $\mathrm{M}_{ex}(\{\cup, \cap, \oplus\})$ we will write $\mathrm{M}_{ex}(\cup, \cap, \oplus)$, for short; similarly for all other problems.

The complexities of our problems strongly depend on the input representation. We assume that natural numbers are given in binary representation and functions are represented by arithmetical circuits with appropriate labels. So, to give an example, a $\{\cup, \oplus, \otimes\}$-function is represented by an arithmetical circuit whose vertices that are not input vertices are labelled with $\cup$, $\oplus$ or $\otimes$. For circuits we require an encoding that permits adjacency tests for two vertices and detection of labels of vertices in deterministic logarithmic space. (A label list and neighbourhood representation by adjacency lists can be assumed.) So, the size of the representation of a circuit is of order the number of vertices and arcs of its underlying graph. It can be verified in nondeterministic logarithmic space whether an input represents an $\mathcal{O}$-function for $\mathcal{O} \subseteq \{\cup, \cap, ^{-}, \oplus, \otimes\}$ (Lemma 5.1). Remember that the upper space complexity bound is mostly required for verifying that the graph underlying the circuit is acyclic. The following observation is easy.

**Lemma 5.2** *Let* $\mathcal{O} \subseteq \mathcal{O}' \subseteq \{\cup, \cap, ^{-}, \oplus, \otimes\}$.

*(1)* $\mathrm{M}_{tm}(\mathcal{O}) \leq_m^{\mathrm{L}} \mathrm{M}_{tm}(\mathcal{O}')$

*(2)* $\mathrm{M}_{ex}(\mathcal{O}) \leq_m^{\mathrm{L}} \mathrm{M}_{ex}(\mathcal{O}')$.

**Proof:** If a given input does not fulfill basic requirements for being an instance of $\mathrm{M}_{tm}(\cup, \cap, ^{-}, \oplus, \otimes)$ or $\mathrm{M}_{ex}(\cup, \cap, ^{-}, \oplus, \otimes)$, it can simply be output by the reducing function. Acyclicness of the intended circuits is not such a basic requirement. If an input "looks" like an instance, it suffices to verify that appearing labels are only from set $\mathcal{O}$, which can certainly be done in logarithmic space.

■

It should be mentioned here that instance verification becomes easy, if additionally a certificate for being acyclic is required for the circuits. Such a certificate can be a topological ordering of the vertices (Theorem 2.2). Then, instance verification can be done in logarithmic space. However, all complexity results that we will prove for membership problems for recurrent systems remain true as they will be stated. In other words, the complexity of instance verification does not have any effect on the complexities of our problems.

As first complexity results for membership problems for recurrent systems, we will prove upper bounds for the general problems. Interestingly, it is an open and difficult question whether $\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$ is computable. We can prove only a weaker result and place the problem in the lower regions of the arithmetical hierarchy. But before, we define a surely decidable variant of $\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$. Instead of the whole set of natural numbers, the variant takes recurrent systems that are defined only over finite intervals of $\mathbb{N}$. Let $a \in \mathbb{N}$. A finite recurrent system $S$ over $\{0, \ldots, a\}$ is defined similar to usual recurrent systems only that functions are defined only over $\{0, \ldots, a\}$. For our operations, we precise as follows. Let $A \subseteq \mathbb{N}$ be a set, and let $B$ and $C$ be integer expressions that are built using operators from the set $\{\cup, \cap, ^-, \oplus, \otimes\}$. Then,

$$\begin{aligned}
[A]_{\leq a} &=_{\text{def}} A \cap \{0, \ldots, a\} \\
[B \cup C]_{\leq a} &=_{\text{def}} [B]_{\leq a} \cup [C]_{\leq a} \\
[B \cap C]_{\leq a} &=_{\text{def}} [B]_{\leq a} \cap [C]_{\leq a} \\
[\overline{B}]_{\leq a} &=_{\text{def}} \overline{([B]_{\leq a})} \cap \{0, \ldots, a\} \\
[B \oplus C]_{\leq a} &=_{\text{def}} ([B]_{\leq a} \oplus [C]_{\leq a}) \cap \{0, \ldots, a\} \\
[B \otimes C]_{\leq a} &=_{\text{def}} ([B]_{\leq a} \otimes [C]_{\leq a}) \cap \{0, \ldots, a\} .
\end{aligned}$$

So, $\{\cup, \cap, ^-, \oplus, \otimes\}$-functions over $\{0, \ldots, a\}$ compute subsets of $\{0, \ldots, a\}$, especially at every vertex of the representing circuits. Let $\mathrm{aM}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$ be the set of tuples $(S, a, t, b)$ where $S$ is a recurrent system, $a$, $t$ and $b$ are natural numbers given in binary representation and $b \in [S(t)]_{\leq a}$. Variants of $\mathrm{aM}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$ for recurrent systems with restricted sets of operations are defined in the usual style. We have to emphasise the fact that $b \in [S(t)]_{\leq a}$ is different from $b \in S(t) \cap \{0, \ldots, a\}$. This effect is illustrated by the following example, where we choose $a = b = 1$:

$$1 \in (((\{1,2\} \cap \{2,3\}) \otimes \{0\}) \oplus \{1\}) \cap \{0,1\} = \{1\}$$

but

$$1 \notin \Big(\Big(\Big(([\{1,2\}]_{\leq 1} \cap [\{2,3\}]_{\leq 1}) \otimes [\{0\}]_{\leq 1}\Big) \cap \{0,1\}\Big) \oplus [\{1\}]_{\leq 1}\Big) \cap \{0,1\} = \emptyset .$$

Another example of a bounded circuit evaluation is depicted in Figure 5. The bound is set to $a = 11$. The result is the set of prime numbers not greater than 11, and it is easy to verify that, for every chosen bound, the result is the set of prime numbers not greater than the specified bound.

**Lemma 5.3** $\mathrm{aM}_{tm}(\cup, \cap, ^{-}, \oplus, \otimes)$ *is in* EXP.

**Proof:** Let $(S, a, t, b)$ be an instance of $\mathrm{aM}_{tm}(\cup, \cap, ^{-}, \oplus, \otimes)$ where $S = (\mathcal{F}, A)$ is a recurrent system of dimension $n$, $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$. Let $C_1, \ldots, C_n$ be arithmetical $\{\cup, \cap, ^{-}, \oplus, \otimes\}$-circuits representing the functions $f_1, \ldots, f_n$, respectively. Let finally $\nu$ denote the sum of the numbers of vertices appearing in $C_1, \ldots, C_n$. To obtain $[S(t)]_{\leq a}$, we have to compute $([S_1(i)]_{\leq a}, \ldots, [S_n(i)]_{\leq a})$ for every $i \leq t$. Per iteration step, we have to compute $\nu$ intermediate results, that correspond to the vertices of the circuits. Every single result can be obtained from already computed ones in exponential time (measured in the size of the input), so that all computations of one iteration step can be performed in time

$$\nu \cdot p(\nu \cdot 2^{\log a}) \leq 2^{q(\log \nu + \log a)} = 2^{q(\log \nu \cdot a)}$$

for some polynomials $p$ and $q$. Furthermore, only exponentially many iteration steps have to be perfermed, which makes exponential time in total.

■

We want to show that $\mathrm{aM}_{tm}$ is not an arbitrary modification of $\mathrm{M}_{tm}$ but a usual—a *useful*—one. We need a technical result, that already contains the main property for this topic.

**Lemma 5.4** *Let $n \geq 1$. Let $f = f(x_1, \ldots, x_n)$ be an $n$-ary $\{\cup, \cap, ^{-}, \oplus, \otimes\}$-function, and let $a_1, \ldots, a_n \in \mathbb{N}$. Let $\mathbf{a} =_{\mathrm{def}} (\{a_1\}, \ldots, \{a_n\})$. Then, there is a constant $c \geq 1$ such that, for every $a \geq c$, holds:*

$$[f(\mathbf{a})]_{\leq a} = f(\mathbf{a}) \cap \{0, \ldots a\}$$

*and*

$$\emptyset \subset f(\mathbf{a}) \subset \mathbb{N} \implies \emptyset \subset [f(\mathbf{a})]_{\leq a} \subset \{0, \ldots, a\}.$$

**Proof:** Let $\mathbf{x} =_{\mathrm{def}} (x_1, \ldots, x_n)$. We prove the statement by induction over the construction of $f$. Let $i \in \{1, \ldots, n\}$ such that $f(\mathbf{x}) = x_i$, i.e., $f$ is simple. We set $c =_{\mathrm{def}} a_i + 1$. Then, for every $a \geq c$:

$$[f(\mathbf{a})]_{\leq a} = \{a_i\} \cap \{0, \ldots, a\} = f(\mathbf{a}) \cap \{0, \ldots, a\} = \{a_i\}.$$

Furthermore, since $\emptyset \subset f(\mathbf{a}) \subset \mathbb{N}$ and $\emptyset \subset [f(\mathbf{a})]_{\leq a} \subset \{0, \ldots, a\}$, the statement holds for this case. Note that $c = a_i + 1$ is necessary for the case $a_i = 0$.

Now, let $f$ be composite. Without loss of generality, we can assume that $f$ does not use $\cap$, since this operator can be built from using only $\cup$ and $^{-}$. There are

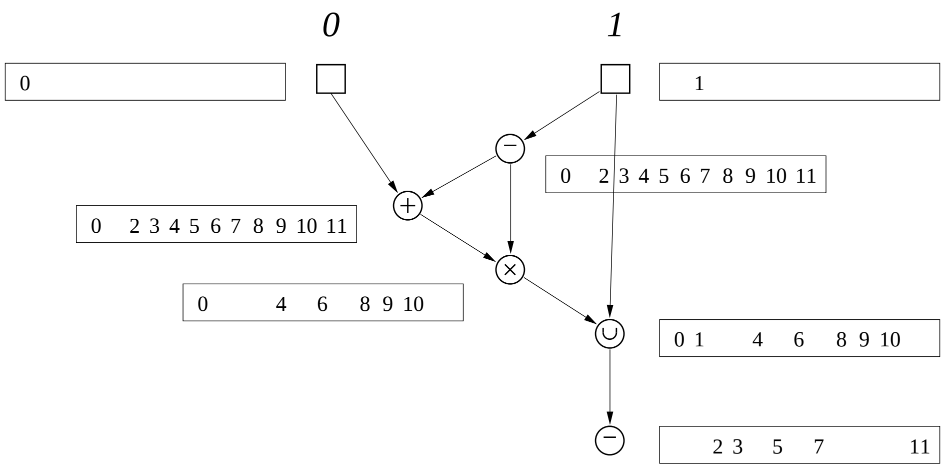


**Figure 5** Depicted is an arithmetical $\{\cup, {}^{-}, \oplus, \otimes\}$-circuit. It is evaluated in the sense of $\mathrm{aM}_{tm}(\cup, {}^{-}, \oplus, \otimes)$ with bound $a = 11$, i.e., only numbers not greater than 11 are taken into account. Input values are 0 and 1, and the results of each vertex are annotated.

$n$-ary functions $f_1$ and $f_2$ fulfilling the assumptions using the constants $c_1$ and $c_2$, respectively. We distinguish four cases.

(a) Let $f(\mathbf{x}) = \overline{f_1(\mathbf{x})}$. We set $c =_{\text{def}} c_1$. We have to verify the properties. Let $a \geq c$.

$$\begin{aligned}[f(\mathbf{a})]_{\leq a} = [(\overline{f_1(\mathbf{a})})]_{\leq a} &= \overline{[f_1(\mathbf{a})]_{\leq a}} \cap \{0,\dots,a\}\\ &= \overline{f_1(\mathbf{a}) \cap \{0,\dots,a\}} \cap \{0,\dots,a\}\\ &= \overline{f_1(\mathbf{a})} \cap \{0,\dots,a\} = f(\mathbf{a}) \cap \{0,\dots,a\}\end{aligned}$$

The second property immediately holds by the definition of the complementation operation.

(b) Let $f(\mathbf{x}) = f_1(\mathbf{x}) \cup f_2(\mathbf{x})$. Applying the induction hypothesis, it holds for every $a \geq \max\{c_1, c_2\}$:

$$\begin{aligned}[f(\mathbf{a})]_{\leq a} = [f_1(\mathbf{a})]_{\leq a} \cup [f_2(\mathbf{a})]_{\leq a} &= (f_1(\mathbf{a}) \cap \{0,\dots,a\}) \cup (f_2(\mathbf{a}) \cap \{0,\dots,a\})\\ &= (f_1(\mathbf{a}) \cup f_2(\mathbf{a})) \cap \{0,\dots,a\}\\ &= f(\mathbf{a}) \cap \{0,\dots a\}\,.\end{aligned}$$

For proving the second property, we have to be more careful. Assume that $\emptyset \subset f(\mathbf{a})$. Then, $\emptyset \subset f_1(\mathbf{a})$ or $\emptyset \subset f_2(\mathbf{a})$, and so $\emptyset \subset [f(\mathbf{a})]_{\leq a}$ for every $a \geq \max\{c_1, c_2\}$ by the assumptions about $f_1$ or $f_2$. If there is a number $b \in \mathbb{N}$ such that $b \notin f(\mathbf{a})$, then $[f(\mathbf{a})]_{\leq a} \neq \{0,\dots,a\}$ for every $a \geq \max\{c_1, c_2, b\}$. We set $c =_{\text{def}} \max\{c_1, c_2, b\}$.

(c) Let $f(\mathbf{x}) = f_1(\mathbf{x}) \oplus f_2(\mathbf{x})$. Applying the induction hypothesis, it holds for every $a \geq \max\{c_1, c_2\}$:

$$\begin{aligned}[f(\mathbf{a})]_{\leq a} &= ([f_1(\mathbf{a})]_{\leq a} \oplus [f_2(\mathbf{a})]_{\leq a}) \cap \{0,\dots,a\}\\ &= ((f_1(\mathbf{a}) \cap \{0,\dots,a\}) \oplus (f_2(\mathbf{a}) \cap \{0,\dots,a\})) \cap \{0,\dots,a\}\\ &= (f_1(\mathbf{a}) \oplus f_2(\mathbf{a})) \cap \{0,\dots,a\} = f(\mathbf{a}) \cap \{0,\dots,a\}\,.\end{aligned}$$

Note that the last but second equality holds due to the monotony property of addition. The definition of the sought constant and the verification of the second property are similar to the previous case. Let $\emptyset \subset f(\mathbf{a})$, i.e., there are numbers $b_1 \in f_1(\mathbf{a})$ and $b_2 \in f_2(\mathbf{a})$ such that $b_1 + b_2 \in f(\mathbf{a})$. Hence, $b_1 + b_2 \in [f(\mathbf{a})]_{\leq a}$ for every $a \geq \max\{c_1, c_2, b_1 + b_2\}$. Similarly, if there is $b \in \mathbb{N}$ such that $b \notin f(\mathbf{a})$, then $b \notin [f(\mathbf{a})]_{\leq a}$ for every $a \geq \max\{c_1, c_2, b\}$. Thus, we set $c =_{\text{def}} \max\{c_1, c_2, b_1 + b_2, b\}$.

(d) Let $f(\mathbf{x}) = f_1(\mathbf{x}) \otimes f_2(\mathbf{x})$. This is the case that causes most difficulties. Similar to the case $\oplus$ we obtain for every $a \geq \max\{c_1, c_2\}$:

$$\begin{aligned}[f(\mathbf{a})]_{\leq a} &= ([f_1(\mathbf{a})]_{\leq a} \otimes [f_2(\mathbf{a})]_{\leq a}) \cap \{0,\dots,a\}\\ &= ((f_1(\mathbf{a}) \cap \{0,\dots,a\}) \otimes (f_2(\mathbf{a}) \cap \{0,\dots,a\})) \cap \{0,\dots,a\}\\ &= (f_1(\mathbf{a}) \otimes f_2(\mathbf{a})) \cap \{0,\dots,a\} = f(\mathbf{a}) \cap \{0,\dots,a\}\,.\end{aligned}$$

It is clear that all equalities hold for most cases but 0. However, by definition, if $0 \in f(\mathbf{a})$, then neither $f_1(\mathbf{a})$ nor $f_2(\mathbf{a})$ are empty and so, by induction hypothesis, neither $f_1(\mathbf{a}) \cap \{0, \ldots, a\}$ nor $f_2(\mathbf{a}) \cap \{0, \ldots, a\}$ are empty for every $a \geq \max\{c_1, c_2\}$. The verification of the second property and the definition of the constant are straightforward and similar to the case $\oplus$.

■

**Theorem 5.5** *Let $n \geq 1$. Let $S$ be a recurrent system, and let $t$ and $b$ be natural numbers. It holds that $b \in S(t)$ if and only if there is $c \geq 1$ such that for all $a \geq c$: $(S, a, t, b) \in \mathrm{aM}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$.*

**Proof:** Without precisely explaining the connection, we can restrict on the situation $t = 1$. The basic idea is to replace the recurrent system by a single function. We will encounter this construction when we will prove polynomial space decidability results (we compute the "circuit representation" of $S(t)$ by $t$-fold superposition). But then, the result follows from Lemma 5.4 using any $a \geq \max\{c, b\}$ where $c$ is the constant of the lemma.

■

The complexity class $\Sigma_2$ of the second level of the arithmetical hierarchy is defined as the class of sets $A$ of the following form: there are a decidable set $B$ and numbers $r, s \geq 0$ such that, for every $x_1, \ldots, x_k \in \Sigma^*$:

$$(x_1, \ldots, x_k) \in A \iff \exists y_1 \ldots \exists y_r \forall z_1 \ldots \forall z_s (x_1, \ldots, x_k, y_1, \ldots, y_r, z_1, \ldots, z_s) \in B\,.$$

Note that the quantifiers are unlimited, i.e., they range over the whole set $\Sigma^*$.

**Corollary 5.6 (Glaßer, [34])**
*$\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus, \otimes)$ and $\mathrm{M}_{ex}(\cup, \cap, ^-, \oplus, \otimes)$ are contained in $\Sigma_2$.*

## 5.4 Membership problems for arithmetical circuits

The study of membership problems has a long tradition. Stockmeyer and Meyer, already in the eighth decade of the past century, considered sets of natural numbers that can be built by using the operators $\cup$, $^-$ and $\oplus$ applied to singleton sets of natural numbers [79]. In our notation, Stockmeyer and Meyer formed sets using arithmetical $\{\cup, ^-, \oplus\}$-circuits, where the underlying graphs are trees. Such circuits were called *formulas* by McKenzie and Wagner [61].

A broader—almost complete—investigation of such membership problems was undertaken by McKenzie and Wagner. The authors studied membership problems for sets that can be defined by $\{\cup, \cap, ^-, \oplus, \otimes\}$-circuits and -formulas. Similar to our membership problems, they restricted the sets of possible operations to arbitrary subsets and obtained completeness results for many complexity classes. To be precise, McKenzie and Wagner defined the following problems. Let $\mathcal{O} \subseteq \{\cup, \cap, ^-, \oplus, \otimes\}$. Then,

$$\mathrm{MC}(\mathcal{O}) =_{\mathrm{def}} \{(C, A, b) : C \text{ is an arithmetical } \mathcal{O}\text{-circuit and } b \in [C(A)]\}$$

where $A$ is a tuple containing natural numbers suitable for circuit $C$, and $[C(A)]$ denotes the set that is defined by circuit $C$ whose input vertices are labelled with the corresponding entries of $A$. Additionally, $C$ is furnished with a topological ordering of the vertices of its underlying graph for efficient input verification. So, $\mathrm{MC}(\mathcal{O})$ can be considered the sets of tuples $(S, t, b)$ in $\mathrm{M}_{tm}(\mathcal{O})$ where $t = 1$. We re-emphasise that $\mathrm{MC}(\mathcal{O})$ requires topological orderings; nevertheless the problems are comparable. In Figure 6, the complexity results for the circuit membership problems from [61] are stated. If lower and upper complexity bounds of a problem coincide, the problem is complete for this class. The complexity class coR is a probabilistic complexity class and a subclass of coNP. It is interesting that it is not known whether $\mathrm{MC}(\cup, \cap, ^-, \oplus, \otimes)$ is decidable. It seems more likely that this is not the case (consider the discussion in [61]).

**Theorem 5.7 (McKenzie and Wagner, [61])**
*The results of Figure 6 hold.*

It is interesting that most complexity classes that appear in Theorem 5.7 are important for our membership problems, too. We will see that, in some cases, even the existential membership problem does not increase the complexity. Particularly, this will hold for $\{\cup, \oplus, \otimes\}$-circuits and recurrent $\{\cup, \oplus, \otimes\}$-systems. This behaviour seems surprising, and the proof will be one of the most involved ones of the following chapter.

| **Operation set** | **Circuit membership problem** MC | |
|---|---|---|
| | Lower bound | Upper bound |
| $\cup\ \cap\ ^{-}\ \oplus\ \otimes$ | NEXP | ? |
| $\cup\ \cap\ \oplus\ \otimes$ | NEXP | |
| $\cup\ \oplus\ \otimes$ | PSPACE | |
| $\cap\ \oplus\ \otimes$ | P | coR |
| $\oplus\ \otimes$ | P | |
| $\cup\ \cap\ ^{-}\ \oplus$ | PSPACE | |
| $\cup\ \cap\ \oplus$ | PSPACE | |
| $\cup\ \oplus$ | NP | |
| $\cap\ \oplus$ | $\mathrm{C_{=}L}$ | |
| $\oplus$ | $\mathrm{C_{=}L}$ | |
| $\cup\ \cap\ ^{-}\ \otimes$ | PSPACE | |
| $\cup\ \cap\ \otimes$ | PSPACE | |
| $\cup\ \otimes$ | NP | |
| $\cap\ \otimes$ | $\mathrm{C_{=}L}$ | P |
| $\otimes$ | NL | |
| $\cup\ \cap\ ^{-}$ | P | |
| $\cup\ \cap$ | P | |
| $\cup$ | NL | |
| $\cap$ | NL | |

**Figure 6** Lower and upper complexity bounds for membership problems for arithmetical $\{\cup, \cap, ^{-}, \oplus, \otimes\}$-circuits. In most cases, if lower and upper bounds are equal, the problems are complete for the specified complexity class. It is not known whether the general membership problem is decidable.

# Chapter 6 Decidable membership problems

Our membership problems for recurrent systems can be partitioned into two sets with respect to decidability. Unlike for membership problems for $\{\cup, \cap, ^-, \oplus, \otimes\}$-circuits, as they were studied by McKenzie and Wagner, we will see that we can prove undecidability of some membership problems. However, these three problems are all existential membership problems, and they will be treated in the next chapter. This chapter mainly deals with decidable problems.

We will obtain a number of completeness results for the complexity classes NL, NP and PSPACE, and each of these complexity classes a section of this chapter is dedicated to. The main results will show upper complexity bounds for membership problems. Hardness in most cases follows by rather easy reductions, even though the proof for showing PSPACE-hardness is not simple. Among the NL-complete problems, the main part is dedicated to $\mathrm{M}_{tm}(\otimes)$, the exact membership problem for recurrent $\{\otimes\}$-systems, and we will use the number-of-walks problem xNoMW(1) for solving it.

In the second section, that is dedicated to problems solvable in nondeterministic polynomial time, we will concentrate on solutions for the problems $\mathrm{M}_{tm}(\oplus)$ and $\mathrm{M}_{tm}(\cap, \oplus)$. The former problem is related to the auxiliary problems xPOM(2) and xNoMW(2). For solving the latter problem, we define a new class of problems for recurrent systems—the emptiness problems. These problems are inspired by similar problems for arithmetical circuits introduced by McKenzie and Wagner. We will see that the problems $\mathrm{M}_{ex}(\oplus)$ and $\mathrm{M}_{ex}(\cap, \oplus)$ are NP-complete.

PSPACE-complete membership problems are concidered in the third section. To name two of the results, we will prove that $\mathrm{M}_{ex}(\cup, \cap)$ is PSPACE-hard and $\mathrm{M}_{ex}(\cup, \oplus, \otimes)$ is polynomial-space decidable. Containment is based on analysing so-called computation trees, that represent elements from the output set of iteration steps of recurrent $\{\cup, \oplus, \otimes\}$-systems. Completeness for this and further problems follows from easy reductions.

The last two sections are dedicated to problems that have not yet been treated adequately. Most of them are PSPACE-hard or decidable.

## 6.1 Nondeterministic logarithmic space acceptable membership problems

The problems that we will investigate in this section are the exact membership problems $\mathrm{M}_{tm}()$, $\mathrm{M}_{tm}(^-)$, $\mathrm{M}_{tm}(\cup)$, $\mathrm{M}_{tm}(\cap)$ and $\mathrm{M}_{tm}(\otimes)$ and the existential membership problems $\mathrm{M}_{ex}()$, $\mathrm{M}_{ex}(^-)$ and $\mathrm{M}_{ex}(\cup)$. Remember that $\mathrm{M}_{tm}()$ and $\mathrm{M}_{ex}()$ mean that functions do not contain any operation. All the mentioned problems are contained in the complexity class NL, and many of them are even NL-complete; the others are contained in L. One might suspect that nondeterministic logarithmic space is mostly needed because of the input representation that we have chosen. As we already mentioned McKenzie and Wagner additionally required a topological ordering of the vertices of the circuits to check for acyclicness in deterministic logarithmic space. In our case, however, this would not lower the upper complexity bound as we will see. At first glance surprisingly, $\mathrm{M}_{tm}(^-)$ can nevertheless be solved in (deterministic) logarithmic space, but this is due to the fact that $\{^-\}$-functions are very easily representable.

**Lemma 6.1** $\mathrm{M}_{tm}(^-)$ *is in* L .

**Proof:** $\{^-\}$-Functions are represented by circuits that are unions of directed trees. Every vertex of such a circuit that is not an input vertex has exactly one predecessor (but may have several successors), and orientations point away from input vertices. (An example is given in Figure 7.) Hence, syntactical correctness of instances of $\mathrm{M}_{tm}(^-)$ can be tested in logarithmic space. Equally in logarithmic space, given an $n$-ary $\{^-\}$-function $f(\mathbf{x})$, $\mathbf{x} = (x_1, \dots, x_n)$, it can be determined index number $j \in \{1, \dots, n\}$ such that $f(\mathbf{x}) = x_j$ or $f(\mathbf{x}) = \overline{x_j}$, which certainly does exist and is unique, and verified which of these cases holds. We say that $f$ depends on input component $j$. Starting from the output vertex, the unique source (i.e., input) vertex of the connected component can be determined in logarithmic space as well as the parity of the length of the path. Even parity means an even number of complementation operations applied to the input component; similarly for odd length parity.

Now, let $(S, t, b)$ be an instance of $\mathrm{M}_{tm}(^-)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{^-\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1, \dots, f_n \rangle$ and $A = (a_1, \dots, a_n)$. Let $\mathbf{x} =_{\mathrm{def}} (x_1, \dots, x_n)$. It holds that $b \in S_i(0)$ if and only if $b = a_i$, $i \in \{1, \dots, n\}$, and $b \in S_i(t'+1)$, $t' \geq 0$, if and only if $b \in S_j(t')$, if $f_i(\mathbf{x}) = x_j$, and $b \notin S_j(t')$, if $f_i(\mathbf{x}) = \overline{x_j}$. Consider the sequences $P_\ell =_{\mathrm{def}} (\nu_0, \dots, \nu_\ell)$ of numbers from $\{1, \dots, n\}$ for $\ell \geq 0$ where $\nu_\ell =_{\mathrm{def}} n$ and $f_{\nu_{i+1}}(\mathbf{x}) = x_{\nu_i}$ or $f_{\nu_{i+1}}(\mathbf{x}) = \overline{x_{\nu_i}}$ for all $i \in \{0, \dots, \ell-1\}$. Consider $P_n$. Observe that $P_n$ is the final part of every sequence $P_{t'}$ for $t' \geq n$, since every function of $S$ depends on its unique input component. Note that there are largest $\alpha, \omega \in \{0, \dots, n\}$, $\alpha > \omega$, such that $\nu_\alpha = \nu_\omega$ in $P_n$. In logarithmic space we can determine $\alpha$ and $\omega$. If $t \leq 3n$, $b \in S(t)$ can be decided straightforwardly. Let $t > 3n$. Note that in logarithmic space we cannot always count till $t$. However, since $(\nu_\omega, \dots, \nu_\alpha)$ is unique, this cycle appears many times in $P_t$, and after passing twice this cycle, the result is the same as before. So, the basic idea is to reduce such cycles in $P_t$. Determine $r \in \{0, \dots, 2\delta-1\}$, $\delta =_{\mathrm{def}} \alpha-\omega$, such that $t \equiv n-\alpha+r \pmod{2\delta}$, which can be done in logarithmic space. It holds that $b \in S(t)$ if and only if $b \in S(n-\alpha+r)$, which can be decided in logarithmic space, since $n - \alpha + r < 3n$.

■

Since we only consider $\leq_m^{\mathrm{L}}$-reducibility, we easily conclude L-completeness of $\mathrm{M}_{tm}(^-)$ from Lemma 6.1. It is clear that this corollary is rather uninteresting, and

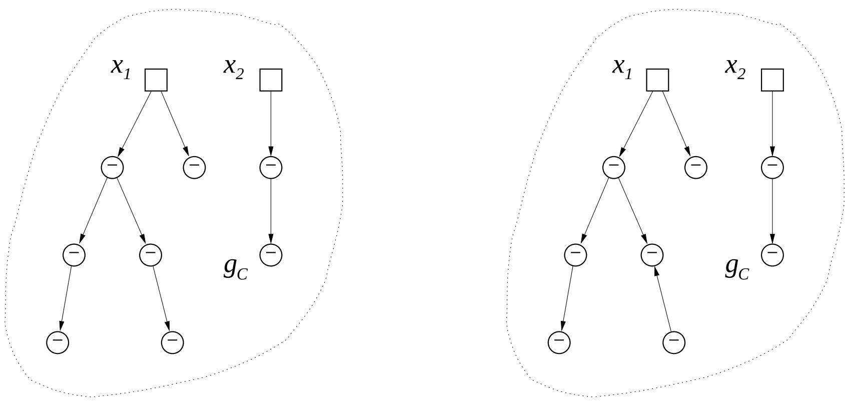


**Figure 7** The left circuit represents the $\{^-\}$-function $f(x_1, x_2) = x_2$ whereas the right structure is not a circuit due to an incorrectly oriented arc. A vertex labelled with $^-$ cannot have two in-coming arcs.

for a more interesting completeness result, we should use stronger reducibilities, for instance based on small circuits (a first overview may be provided by Vollmer's book on circuit complexity [83]).

**Lemma 6.2** *(1)* $\mathrm{M}_{tm}(\cup) \leq_m^{\mathrm{L}} \mathrm{M}_{tm}(\otimes)$.
*(2)* $\mathrm{M}_{tm}(\cap) \leq_m^{\mathrm{L}} \mathrm{M}_{tm}(\otimes)$.

**Proof:** Let $(S,t,b)$ be an instance of $\mathrm{M}_{tm}(\cup)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\cup\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1, \ldots, f_n\rangle$ and $A = (a_1, \ldots, a_n)$. Functions $f_i'$, $i \in \{1, \ldots, n\}$, are obtained from $f_i$ by replacing operation $\cup$ by $\otimes$. Furthermore, for every $i \in \{1, \ldots, n\}$, $a_i' =_{\mathrm{def}} 0$, if $a_i = b$, and $a_i' =_{\mathrm{def}} 1$, if $a_i \neq b$. Then, $S' =_{\mathrm{def}} (\mathcal{F}', A')$ where $\mathcal{F}' =_{\mathrm{def}} \langle f_1', \ldots, f_n'\rangle$ and $A' =_{\mathrm{def}} (a_1', \ldots, a_n')$ is a recurrent $\{\otimes\}$-system. We will show by induction that $b \in S_i(k)$ for every $i \in \{1, \ldots, n\}$ and $k \geq 0$ if and only if $0 \in S_i'(k)$. By definition of $A'$, the claim is true for $k = 0$ and all $i \in \{1, \ldots, n\}$. Let $k \geq 0$, and let $i \in \{1, \ldots, n\}$. Let $C_i'$ be the arithmetical $\{\otimes\}$-circuit that represents $\{\otimes\}$-function $f_i'$, and let $C_i$ be the arithmetical $\{\cup\}$-circuit that represents $f_i$. Then, $0 \in S_i'(k+1)$ if and only if there is $s \in \{1, \ldots, n\}$ such that the $s$-th input vertex of $C_i'$ is start vertex of a path to the output vertex of $C_i'$ and $0 \in S_s'(k)$, and this holds if and only if there is $r \in \{1, \ldots, n\}$ such that the $r$-th input vertex of $C_i$ is start vertex of a path to the output vertex of $C_i$ and $b \in S_r(k)$. By construction of $S'$, $r$ and $s$ can be chosen equal. Hence, $0 \in S_i'(k+1)$ if and only if $b \in S_i(k+1)$, and $(S', t, 0) \in \mathrm{M}_{tm}(\otimes)$ if and only if $(S,t,b) \in \mathrm{M}_{tm}(\cup)$.

In case of $\mathrm{M}_{tm}(\cap)$, we can apply a similar reduction. We only have to make the following modification. Let $(S,t,b)$ be an instance of $\mathrm{M}_{tm}(\cap)$ where $S = (\mathcal{F}, A)$ and $A = (a_1, \ldots, a_n)$, $n = \dim S$. For every $i \in \{1, \ldots, n\}$, $a_i'' =_{\mathrm{def}} 1$, if $a_i = b$, and $a_i'' =_{\mathrm{def}} 0$, if $a_i \neq b$. Let $A'' =_{\mathrm{def}} (a_1'', \ldots, a_n'')$; $\mathcal{F}''$ is obtained from $\mathcal{F}$ as described above, where intersection operations are replaced by $\otimes$. Let $S'' =_{\mathrm{def}} (\mathcal{F}'', A'')$. Then, $(S'', t, 1) \in \mathrm{M}_{tm}(\otimes)$ if and only if $(S,t,b) \in \mathrm{M}_{tm}(\cap)$ by a proof similar to the one above.

■

For showing that $\mathrm{M}_{tm}(\otimes)$ is contained in NL, we give a logarithmic-space algorithm that has access to an NL-set oracle. This oracle set will be xNoMW(1), the problem asking for the exact number of marked walks from one to another specified vertex where the queried number is given in unary representation. In addition, we have to compute products of natural numbers in small space. By *iterated multiplication*, we mean the computation of the product of, possibly more than two, given numbers. This problem defines a function that we have called ItMult. Remember that ItMult is an FL-function (Theorem 3.6).

Before we give the proof of $\mathrm{M}_{tm}(\otimes) \in \mathrm{NL}$, we discuss the main ideas of the proof on a small example. Consider the $\{\otimes\}$-circuit in Figure 8. The depicted circuit represents the function $f(x_1, x_2, x_3) = (x_1)^3 \cdot (x_2)^2 \cdot x_3$. Observe that the powers of the variables correspond to the numbers of paths from the input vertices to the output vertex $g_C$; for instance, there are (exactly) three $x_1, g_C$-paths. So, for evaluating $\{\otimes\}$-circuits, determining the numbers of paths from input vertices to the output vertex is important. This idea has been used by McKenzie and Wagner to show that MC($\otimes$) is in NL ([61] and Theorem 5.7). Our proof for showing that $\mathrm{M}_{tm}(\otimes)$ is in NL extends this idea.

**Lemma 6.3** $\mathrm{M}_{tm}(\otimes)$ *is in* NL *.*

**Proof:** We will show that $\mathrm{M}_{tm}(\otimes)$ can be decided in logarithmic space with access to oracle xNoMW(1). Let $(S, t, b)$ be an instance of $\mathrm{M}_{tm}(\otimes)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\otimes\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1, \ldots, f_n\rangle$ and $A = (a_1, \ldots, a_n)$. Let $C_1, \ldots, C_n$ be the arithmetical $\{\otimes\}$-circuits representing $f_1, \ldots, f_n$, respectively. We obtain a directed graph $C$ from $C_1, \ldots, C_n$ in the following way: glue together the input vertices corresponding to the same input component (i.e., glue together the vertices in $C_1, \ldots, C_n$ labelled 1, glue together the vertices in $C_1, \ldots, C_n$ labelled 2, and so on), and glue together every output vertex with the input vertex corresponding to its circuit number (e.g., output vertex of $C_1$ is glued together with input vertex labelled 1). If there is an output vertex that is also an input vertex, an arc is set instead, i.e., if, for instance, the input vertex of $C_1$ labelled with 3 is also the output vertex of $C_1$, an arc from input vertex 3 to input vertex 1 is set in $C$. Let $M$ be the set of vertices of $C$ that are labelled input vertices. Let $z_1, \ldots, z_n$ be the vertices in $M$, that we call marked, and $z_i$ is the input vertex labelled $i$. Note that, for every pair $i, j$ of numbers from $\{1, \ldots, n\}$, the number of 2-marked $z_i, z_j$-walks in $C$ is exactly the number of paths in $C_j$ from input vertex labelled $i$ to the output vertex.

Now, let $c_1, \ldots, c_n$ be natural numbers such that, for every $i \in \{1, \ldots, n\}$, $c_i$ is the number of $(t+1)$-marked $z_i, z_n$-walks in $C$. It holds that $b \in S(t)$ if and only if $b = \prod_{i=1}^{n}(a_i)^{c_i}$. We assume $0^0 = 1$. For verification of this condition, we distinguish three cases:

(A) If $b = 0$, it suffices to find $i \in \{1, \ldots, n\}$ such that $a_i = 0$ and $c_i \neq 0$.

(B) If $b = 1$, it suffices to verify that $c_i = 0$ holds for all $i \in \{1, \ldots, n\}$ for which $a_i \neq 1$ does hold. Remember that, by definition of circuits, there must be $r \in \{1, \ldots, n\}$ such that $c_r > 0$.

(C) If $b \geq 2$, it must be verified that, for all $i \in \{1, \ldots, n\}$, $a_i = 0$ implies $c_i = 0$, and $c_i \leq \log b$, if $a_i \geq 2$. Then, $c_i$ for $a_i \neq 0$ are determined, and by iterated multiplication the value of $S(t)$ can be computed.

All three cases can be handled in logarithmic space using xNoMW(1) as oracle. Case (C) strongly depends on the fact that iterated multiplication is an FL-function due to Theorem 3.6. Since xNoMW(1) is in NL due to Corollary 4.12, $\mathrm{M}_{tm}(\otimes) \in \mathrm{L}^{\mathrm{NL}} = \mathrm{NL}$.

■

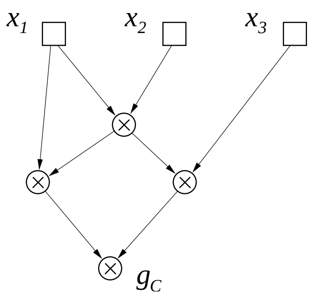


**Figure 8** A $\{\otimes\}$-circuit representing the function $f(x_1, x_2, x_3) = (x_1)^3(x_2)^2x_3$ .

Note that cases (A) and (B) of the last proof solve the problems $\mathrm{M}_{tm}(\cup)$ and $\mathrm{M}_{tm}(\cap)$, respectively, after application of the reductions defined in the proof of Lemma 6.2.

Now, we can show completeness for three exact membership problems.

**Theorem 6.4** $\mathrm{M}_{tm}(\cup)$, $\mathrm{M}_{tm}(\cap)$ *and* $\mathrm{M}_{tm}(\otimes)$ *are* NL-*complete.*

**Proof:** Containment in NL of all these problems is due to Lemmata 6.2 and 6.3. For hardness, we reduce ACYCGAP to $\mathrm{M}_{tm}(\cup)$ and ACYCGIP to $\mathrm{M}_{tm}(\cap)$, which establishes the desired results. Let $(G, u, v)$ be an instance of ACYCGAP where $G = (V, A)$ is a directed graph and $u$ and $v$ are vertices of $G$. We have to decide whether $G$ is acyclic and contains a $u, v$-path. Let $w_1$ and $w_2$ be new vertices. We obtain $H$ from $G$ by adding $w_1$ and $w_2$ to $G$ and arcs from $w_1$ to $u$ and from $v$ to $w_2$. Observe that $G$ has a $u, v$-path if and only if $H$ has a $w_1, w_2$-path. We convert $H$ into a graph whose vertices all have at most two in-coming arcs: if $x$ is a vertex of $H$ that is endpoint of $r \geq 3$ arcs, replace $x$ by $r - 1$ new vertices by application of the method executed on a small example in Figure 9; let $H'$ be the resulting graph. Then, it holds that $(G, u, v) \in$ ACYCGAP if and only if $(H', w_1, w_2) \in$ ACYCGAP. Now, let $z$ be a new vertex. We add $z$ to $H'$ and arcs from $z$ to every vertex with exactly one in-coming arc; we obtain $H''$. Then, $(H', w_1, w_2) \in$ ACYCGAP if and only if $(H'', w_1, w_2) \in$ ACYCGAP.

From $H''$ we construct a $\{\cup\}$-circuit $C$. Let $s$ be the number of vertices of $H''$ without any in-coming arcs. By construction, $s \geq 2$. Every vertex of $H''$ with in-coming arcs is labelled with $\cup$, and every vertex without in-coming arcs is labelled input vertex, where $w_1$ shall be the first. The output vertex of $C$ is $w_2$. For constructing a recurrent system, let $f(x_1, \ldots, x_s)$ be the $\{\cup\}$-function represented by $C$. We define the functions of our recurrent $\{\cup\}$-system as follows. The system will have dimension $s + 1$. (We use this inefficient construction, since we want to reuse it later.) For every $i \in \{1, \ldots, s\}$, let $f_i(x_1, \ldots, x_{s+1}) =_{\text{def}} x_i$. Furthermore, let $f_{s+1}(x_1, \ldots, x_{s+1}) =_{\text{def}} f(x_1, \ldots, x_s)$. Let $\mathcal{F} =_{\text{def}} \langle f_1, \ldots, f_{s+1} \rangle$, $A =_{\text{def}} (1, 0, \ldots, 0)$ and $S =_{\text{def}} (\mathcal{F}, A)$. Then, $1 \in S(1)$ if and only if $(H'', w_1, w_2) \in$ ACYCGAP, i.e., if and only if $H''$ is acyclic and contains a $w_1, w_2$-path. Hence, $(G, u, v) \in$ ACYCGAP if and only if $(S, 1, 1) \in \mathrm{M}_{tm}(\cup)$.

The reduction from ACYCGIP to $\mathrm{M}_{tm}(\cap)$ uses the same reduction with two slight modifications: $S'$ emerges from $S$ by replacing every $\cup$ by $\cap$, and $(G, u, v) \in$ ACYCGIP if and only if $(S', 1, 0) \in \mathrm{M}_{tm}(\cap)$, since a $u, v$-path in $G$ would imply $S'(1) = \emptyset$ or $S'(1) = \{1\}$.

■

Interestingly, only one existential membership problem turns out to be complete for NL. The two others are far more complicated (under reasonable assumptions such as NL $\subset$ NP).

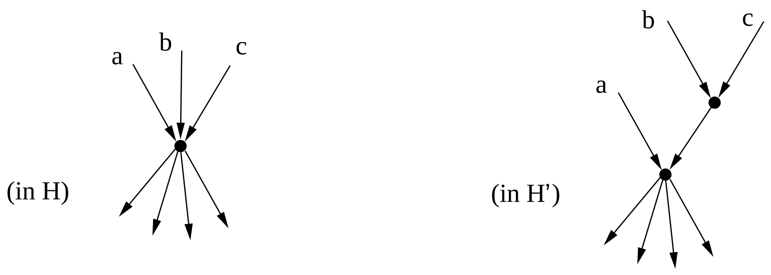


**Figure 9** Example for the construction in the proof of Theorem 6.4. The left situation is replaced by the right one.

**Theorem 6.5** *(1)* $\mathrm{M}_{ex}(^{-})$ *is in* L.

*(2)* $\mathrm{M}_{ex}(\cup)$ *is* NL*-complete.*

**Proof:** For statement (1), let $(S, b)$ be an instance of $\mathrm{M}_{ex}(^{-})$, $n =_{\mathrm{def}} \dim S$. The proof of Lemma 6.1 already shows that $b \in [S]$ if and only if $b \in S(t)$ for some $t \leq 3n$. In deterministic logarithmic space, it can be tested whether one of $(S, 0, b)$, $(S, 1, b)$, …, $(S, 3n, b)$ is contained in $\mathrm{M}_{tm}(^{-})$ using $\mathrm{M}_{tm}(^{-})$ as oracle. Hence $\mathrm{M}_{ex}(^{-}) \in \mathrm{L}^{\mathrm{L}} = \mathrm{L}$.

For statement (2), let $(S, b)$ be an instance of $\mathrm{M}_{ex}(\cup)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\cup\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ and $A = (a_1, \ldots, a_n)$. We show that $b \in [S]$ if and only if $b \in S(0) \cup \cdots \cup S(n-1)$. The proof is based on the construction of Lemma 6.3. We say that $f_i$ *depends on input* $j$, $i, j \in \{1, \ldots, n\}$, if, for every $A_1, \ldots, A_n \subseteq \mathbb{N}$, $0 \in f_i(A_1, \ldots, A_{i-1}, \{0\}, A_{i+1}, \ldots, A_n)$. In other words, $f_i$ depends on input $j$, if the circuit representing $f_i$ contains a path from input vertex labelled $j$ to the output vertex. We construct an auxiliary graph $G$ as follows: $G$ contains $n$ vertices named $1, \ldots, n$, and there is an arc from vertex $j$ to vertex $i$ if and only if $f_i$ depends on input $j$. Note that $G$ is a reduced version of graph $C$ of Lemma 6.3, where we only keep information about accessibility but not about the number of paths. Let $a_r = b$ for some $r \in \{1, \ldots, n\}$. It holds that $b \in S(t)$ if and only if there is an $r, n$-walk in $G$ of length $t$, i.e., containing $t + 1$ vertices. Let $P = (u_0, \ldots, u_t)$ be an $r, n$-walk of length $t$ in $G$. By the pigeon hole principle, there is also an $r, n$-path in $G$ that contains at most $n$ vertices, which proves that $\mathrm{M}_{ex}(\cup)$ is in $\mathrm{L}^{\mathrm{NL}} = \mathrm{NL}$ with an argument similar to statement (1).

For hardness, we reduce ACYCGAP to $\mathrm{M}_{ex}(\cup)$. Let $S = (\mathcal{F}, A)$ be the $(s+1)$-dimensional recurrent $\{\cup\}$-system defined in the proof of Theorem 6.4. Let $S = (\mathcal{F}, A)$ and $\mathcal{F} = \langle f_1, \ldots, f_{s+1} \rangle$. Let $f_1'(x_1, \ldots, x_{s+1}) =_{\mathrm{def}} x_2$. Then, let $\mathcal{F}' =_{\mathrm{def}} \langle f_1', f_2, \ldots, f_{s+1} \rangle$. Let $S' =_{\mathrm{def}} (\mathcal{F}', A)$. Note that $S_i'(t) = \{0\}$ for every $i \in \{1, \ldots, s\}$ and $t \geq 1$. Then, $(G, u, v) \in$ ACYCGAP if and only if $(S, 1, 1) \in \mathrm{M}_{tm}(\cup)$ if and only if $(S', 1) \in \mathrm{M}_{ex}(\cup)$.

■

## 6.2 Nondeterministic polynomial time acceptable membership problems

Mainly, we investigate recurrent $\{\cap, \oplus\}$-systems and the complexities of the corresponding membership problems. As a by-product we also determine the complexity of the existential membership problem for recurrent $\{\otimes\}$-systems. Furthermore, we introduce the *emptiness* problem for recurrent systems, that asks whether, in some iteration step, the empty set is the result. Our first considerations, however, are made even for recurrent $\{\cap, \oplus, \otimes\}$-systems.

An immediate property of $\{\cap, \oplus, \otimes\}$-functions is that they can compute sets of cardinality at most 1, if they are applied to singleton sets only. Since the result may also be empty, multiplication with 0 can be considered an emptiness test. And deciding emptiness will be a major problem to solve in this context. In contrast, multiplication with 0 for $\{\cup, \oplus, \otimes\}$-functions is nothing else than an accessibility problem, since such functions cannot compute empty sets.

First, we want to understand $\{\cap, \oplus, \otimes\}$-functions. The intersection operation in this context can be regarded as an equivalence test. With this notion in mind we can rewrite $\{\cap, \oplus, \otimes\}$-functions by systems of $\{\oplus, \otimes\}$-functions in a specific sense that is precised in the following lemma. Note that a $\{\oplus, \otimes\}$-function is a polynomial in several variables with natural coefficients. A $\{\oplus, \otimes\}$-function over $\mathbb{Z}$ is a polynomial in several variables with integer coefficients. Since every set involved in this section is of cardinality at most 1, we will avoid cumbersome set braces. The meaning of every term will always be clear from the context.

**Lemma 6.6** *Let $\mathcal{O} \subseteq \{\oplus, \otimes\}$, and let $f = f(x_1, \ldots, x_k)$ be a $k$-ary $(\{\cap\} \cup \mathcal{O})$-function. Then, there are a $k$-ary $\mathcal{O}$-function $f'$ and $k$-ary $\mathcal{O}$-functions $g_1, \ldots, g_r$ over $\mathbb{Z}$, $r \in \mathbb{N}$, such that, for all $\mathbf{a} \in \mathbb{N}^k$,*

$$\begin{aligned} g_i(\mathbf{a}) = 0 \text{ for all } i \in \{1, \ldots, r\} \quad &\Longrightarrow \quad f(\mathbf{a}) = f'(\mathbf{a}) \\ g_i(\mathbf{a}) \neq 0 \text{ for some } i \in \{1, \ldots, r\} \quad &\Longrightarrow \quad f(\mathbf{a}) = \emptyset\,. \end{aligned}$$

**Proof:** We prove the claim by induction. If $f(x_1, \ldots, x_k) = x_j$ for $j \in \{1, \ldots, k\}$ then $f$ fulfills the statement using $f' =_{\text{def}} f$ and $r =_{\text{def}} 0$. Let $f_1$ and $f_2$ be $(\{\cap\} \cup \mathcal{O})$-functions such that $f = f_1 \cap f_2$ or $f = f_1 \oplus f_2$ or $f = f_1 \otimes f_2$. By induction hypothesis, there are $k$-ary $\mathcal{O}$-functions $f_1'$, $f_2'$, $g_1^1, \ldots, g_{r_1}^1$ and $g_1^2, \ldots, g_{r_2}^2$, $r_1, r_2 \geq 0$, such that, for all $\mathbf{a} \in \mathbb{N}^k$,

$$\begin{aligned} g_i^1(\mathbf{a}) = 0 \text{ for all } i \in \{1, \ldots, r_1\} \quad &\Longrightarrow \quad f_1(\mathbf{a}) = f_1'(\mathbf{a}) \\ g_i^2(\mathbf{a}) = 0 \text{ for all } i \in \{1, \ldots, r_2\} \quad &\Longrightarrow \quad f_2(\mathbf{a}) = f_2'(\mathbf{a}) \\ g_i^1(\mathbf{a}) \neq 0 \text{ for some } i \in \{1, \ldots, r_1\} \quad &\Longrightarrow \quad f_1(\mathbf{a}) = \emptyset \\ g_i^2(\mathbf{a}) \neq 0 \text{ for some } i \in \{1, \ldots, r_2\} \quad &\Longrightarrow \quad f_2(\mathbf{a}) = \emptyset\,. \end{aligned}$$

If $f = f_1 \oplus f_2$ or $f = f_1 \otimes f_2$ then $f$ has the claimed property using $f_1' \oplus f_2'$ or $f_1' \otimes f_2'$, respectively, and $g_1^1, \ldots, g_{r_1}^1, g_1^2, \ldots, g_{r_2}^2$. So, let $f = f_1 \cap f_2$. Let $f' =_{\text{def}} f_1'$, $g_i =_{\text{def}} g_i^1$ for $i \in \{1, \ldots, r_1\}$, $g_{r_1+i} =_{\text{def}} g_i^2$ for $i \in \{1, \ldots, r_2\}$ and $g_r =_{\text{def}} f_1' - f_2'$, $r =_{\text{def}} r_1 + r_2 + 1$. We prove correctness of the construction. Let $g_i(\mathbf{a}) = 0$ for $\mathbf{a} \in \mathbb{N}^k$ and all $i \in \{1, \ldots, r\}$. Then $f_1(\mathbf{a}) \neq \emptyset$, $f_2(\mathbf{a}) \neq \emptyset$, $f_1(\mathbf{a}) = f_1'(\mathbf{a})$, $f_2(\mathbf{a}) = f_2'(\mathbf{a})$ and $f_1(\mathbf{a}) - f_2(\mathbf{a}) = 0$, hence $f(\mathbf{a}) = f_1(\mathbf{a}) = f_2(\mathbf{a}) = f'(\mathbf{a})$. If there is $i \in \{1, \ldots, r\}$ such that $g_i(\mathbf{a}) \neq 0$ for $\mathbf{a} \in \mathbb{N}^k$ then $f_1(\mathbf{a}) = \emptyset$ or $f_2(\mathbf{a}) = \emptyset$ or $f_1(\mathbf{a}) \neq f_2(\mathbf{a})$. Hence, $f(\mathbf{a}) = \emptyset$.

■

Since we assume functions to be represented by arithmetical circuits, Lemma 6.6 provides a simple polynomial-time algorithm that reduces $\{\cap, \oplus, \otimes\}$-functions to $\{\oplus, \otimes\}$-functions by simply deleting all $\cap$-vertices in the circuits. Precisely, if $v$

is a $\cap$-labelled vertex in such a circuit then we delete $v$ and connect one predecessor of $v$ to all successors of $v$. Finally, all vertices from which the output vertex cannot be reached are deleted. The function complexity class FP contains all functions that can be computed in polynomial time by Turing transducers.

**Corollary 6.7** *Let $\mathcal{O} \subseteq \{\oplus, \otimes\}$. Let $S$ be a recurrent $(\{\cap\} \cup \mathcal{O})$-system. Then, there is a recurrent $\mathcal{O}$-system $S'$ of dimension* $\dim S$ *such that, for every $t \in \mathbb{N}$, $S(t) \neq \emptyset$ implies $S(t) = S'(t)$, and $S'$ can be computed by an* FP*-function.*

**Proof:** Let $S = (\mathcal{F}, A)$. Let $n =_{\text{def}} \dim S$ and $\mathcal{F} = \langle f_1, \dots, f_n \rangle$. Let $\mathcal{F}' =_{\text{def}} \langle f'_1, \dots, f'_n \rangle$ where $f'_i$, $i \in \{1, \dots, n\}$, is the function corresponding to $f_i$ according to Lemma 6.6. The functions in $\mathcal{F}'$ can be computed in polynomial time, so that $S' =_{\text{def}} (\mathcal{F}', A)$ can be computed by a polynomial-time function. Induction over $t$ shows that $f_i(t) \neq \emptyset$ implies $f_i(t) = f'_i(t)$, $i \in \{1, \dots, n\}$.

■

Corollary 6.7 provides a possible way to handle $\mathrm{M}_{tm}(\cap, \oplus)$: We will give a complexity upper bound for $\mathrm{M}_{tm}(\oplus)$ and show how to decide whether the result of the reduced recurrent system $S'$ can be trusted. The latter of the two tasks requires the study of an appropriate emptiness problem, which becomes complicated and will be solved here only partially. The former task, however, is closely related to exact power-of-matrix and number-of-walks problems and has, therefore, been solved in large parts.

**Lemma 6.8** *(1)* $\mathrm{xPOM}(2) \leq^{\mathrm{L}}_{m} \mathrm{M}_{tm}(\oplus)$.
*(2)* $\mathrm{M}_{tm}(\oplus) \leq^{\mathrm{L}}_{m} \mathrm{xNoMW}(2)$.

**Proof:** (1) Let $(M, k, a, i, j)$ be an instance of xPOM(2). Let $n$ be the dimension of $M$, i.e., the number of columns. Let $\mathbf{a}_1, \dots, \mathbf{a}_n$ be row vectors over $\mathbb{N}$ such that $M^T = (\mathbf{a}_1 \ \cdots \ \mathbf{a}_n)$, where $M^T$ denotes the transpose of $M$. By a construction that is basically the same as used in the proof of Lemma 4.6, we can construct an acyclic graph with $n$ source vertices for every $\mathbf{a}_i$, $i \in \{1, \dots, n\}$, such that the numbers of paths from source vertices to the sink vertex (the designated output vertex of the derived arithmetical circuit) correspond to the entries of $\mathbf{a}_i$. Furthermore, the construction yields a graph where vertices have either zero or two in-coming arcs. By the way, we do not have to make the lengths of all input-vertex-output-vertex paths equal. Then, we label all vertices with two in-coming arcs with $\oplus$ and obtain arithmetical $\{\oplus\}$-circuits $C_1, \dots, C_n$ such that, for every $\mathbf{a} \in \mathbb{N}^n$, $f_i(\mathbf{a}) = \mathbf{a}_i \cdot \mathbf{a}$ for every $i \in \{1, \dots, n\}$, where $f_i$ denotes the function defined by circuit $C_i$. Note that $C_1, \dots, C_n$ can be constructed using only logarithmic space. It then holds that, for every $\mathbf{a} \in \mathbb{N}^n$, $M \cdot \mathbf{a} = (f_1(\mathbf{a}) \ \cdots \ f_n(\mathbf{a}))^T = (\mathbf{a}_1\mathbf{a} \ \cdots \ \mathbf{a}_n\mathbf{a})^T$. Now, let $S = (\mathcal{F}, A)$ be the recurrent $\{\oplus\}$-system where $\mathcal{F} =_{\text{def}} \langle f_1, \dots, f_n \rangle$, represented by the circuits $C_1, \dots, C_n$, respectively, and $A \in \{0,1\}^n$ containing 1 exactly in the $j$-th component. It follows that $M^k \cdot A = ((M^k)_{1j} \ \cdots \ (M^k)_{nj})^T = (S_1(k) \ \cdots \ S_n(k))^T$.

So, it remains to re-index the functions and inputs such that $f_i$ becomes the output function of $S$. Hence, $(M, k, a, i, j) \in \text{xPOM}(2)$ if and only if $(S, k, a) \in \mathrm{M}_{tm}(\oplus)$.

(2) Let $x$ be an input word. In logarithmic space, we can check whether $x$ is of the form such that it may represent an instance of $\mathrm{M}_{tm}(\oplus)$. So, we can identify components of the possible input triple that represent $t$ and $b$ of instances of $\mathrm{M}_{tm}(\oplus)$. We can also identify structures $C_1, \ldots, C_n$ that look like arithmetical $\{\oplus\}$-circuits and a tuple $A = (a_1, \ldots, a_n)$, where $n$ should be the dimension of the contained recurrent system $S$. So, it only remains uncertainty about the acyclicness of the graphs $G_1, \ldots, G_n$ underlying the possible circuits $C_1, \ldots, C_n$, respectively. So, we have to verify two properties: $G_1, \ldots, G_n$ are acyclic, and, if so, $S(t) = b$. For the following construction, we treat these questions separately.

[We assume that $G_1, \ldots, G_n$ all are acyclic. So, $x = (S, t, b)$ is an instance of $\mathrm{M}_{tm}(\oplus)$.]
We construct a marked graph $H_1$ and identify two vertices $s$ and $g$ such that $H_1$ has exactly $b$ $(t+1)$-marked $s, g$-walks if and only if $(S, t, b)$ is in $\mathrm{M}_{tm}(\oplus)$. We obtain subgraph $H_1'$ of $H_1$ from the graphs $G_1, \ldots, G_n$ in three steps:

(a) obtain the first intermediate graph as the disjoint union of the graphs $G_1, \ldots, G_n$,

(b) obtain the second intermediate graph by glueing together the vertices of the graphs $G_1, \ldots, G_n$ that are labelled in $C_1, \ldots, C_n$ with the same numbers, i.e., glue together the $n$ vertices labelled as input vertex 1, glue together the $n$ vertices labelled as input vertex 2, and so on, and label these glued vertices as marked,

(c) obtain the final graph by adding arcs from the output vertex of $C_i$ to the vertex that is the result of glueing together all input vertices labelled with $i$ for all $i \in \{1, \ldots, n\}$.

Remember that we used a similar construction in the proof of Lemma 6.3. Let $M$ denote the set of marked vertices of $H_1'$, and let them be $x_1, \ldots, x_n$, where $x_i$ corresponds to the marked vertex obtained from glueing together all $i$-labelled vertices. Now, observe that, for all pairs $i, j$ of numbers from $\{1, \ldots, n\}$, the number of 2-marked $x_i, x_j$-walks in $H_1'$ is equal to the number of paths in $G_j$ from input vertex $i$ of $C_j$ to the output vertex of $C_j$.

A second subgraph $H_1''$ of $H_1$ is obtained from the numbers of $A$ by the following construction. Let $H_1''$ be an acyclic graph on at least $n+1$ vertices, where we tag vertices $s, s_1, \ldots, s_n$, such that $H_1''$ has exactly $a_i$ $s, s_i$-paths for every $i \in \{1, \ldots, n\}$. Without loss of generality, we can assume that $s$ is a source of $H_1''$ and $s_1, \ldots, s_n$ are sinks. The construction of $H_1''$ is basically equal to the one used in the proof of Lemma 4.6 as well as in the proof of statement (1). Now, obtain $H_1$ from $H_1'$ and $H_1''$ as the disjoint union of $H_1'$ and $H_1''$ and by adding arcs from $s_i$ to $x_i$ for every $i \in \{1, \ldots, n\}$. By a straightforward induction, it can be shown that the number of $(t'+1)$-marked $s, x_i$-walks in $H_1$ is equal to $S_i(t')$ for every $t' \geq 0$ and $i \in \{1, \ldots, n\}$, from which the claim about $H_1$ follows.

[Verifying that $G_1, \ldots, G_n$ all are acyclic.]
We construct a graph $H_2$ from $G_1, \ldots, G_n$ and identify two vertices $u, v$ of $H_2$ such that $H_2$ has exactly one $u, v$-walk, if $G_1, \ldots, G_n$ all are acyclic, and infinitely many $u, v$-walks, if some of the graphs $G_1, \ldots, G_n$ is not acyclic. Let $\nu_1, \ldots, \nu_n$ denote the numbers of vertices of $G_1, \ldots, G_n$, respectively. Let $G_1^1, \ldots, G_1^{\nu_1}, G_2^2, \ldots, G_n^{\nu_n}$ be copies of $G_1, \ldots, G_n$. We obtain $H_2$ first as the disjoint union of $G_1^1, \ldots, G_n^{\nu_n}$ and second by adding the following arcs: an arc from vertex 1 of $G_1^1$ to vertex 2 of $G_1^2$, an arc from vertex 2 of $G_1^2$ to vertex 3 of $G_1^3$, and so on, generally, an arc from vertex $i$ of $G_j^i$ to vertex $i+1$ of $G_j^{i+1}$ for every $j \in \{1, \ldots, n\}$ and every $i \in \{1, \ldots, \nu_j - 1\}$, and then add an arc from vertex $\nu_j$ of $G_j^{\nu_j}$ to vertex 1 of $G_{j+1}^1$ for every $j \in \{1, \ldots, n-1\}$. We address the vertices of $H_2$ by triples of the form $(\alpha, \beta, \gamma)$ which means vertex $\alpha$ of graph $G_\gamma^\beta$. We add a loop to vertex $(1,1,1)$ and mark this vertex. Then, it is the case that there is exactly one $(t+1)$-marked $(1,1,1), (\nu_n, \nu_n, n)$-walk in $H_2$, if $G_1, \ldots, G_n$ all are acyclic. Otherwise, i.e., if there is $j \in \{1, \ldots, n\}$ such that $G_j$ is not acyclic, $H_2$ contains infinitely many such walks, since a cycle contained in $G_j$ can be put arbitrarily many often into every $(1,1,1), (\nu_n, \nu_n, n)$-walk of $H_2$.

[Concluding the construction by combining the two constructed graphs.]
Finally, we obtain graph $H$ as the result of the following two operations: first, obtain the disjoint union of $H_1$ and $H_2$, and second, let $\mathfrak{a}$ and $\mathfrak{e}$ be new vertices, add them to $H$ and add arcs from $\mathfrak{a}$ to $s$ of $H_1$ and $(1,1,1)$ of $H_2$ and arcs from $x_n$ of $H_1$ and $(\nu_n, \nu_n, n)$ of $H_2$ to $\mathfrak{e}$. By the discussions and results above, $H$ contains exactly $b+1$ $(t+1)$-marked $\mathfrak{a}, \mathfrak{e}$-walks if and only if $(S, t, b) \in \mathrm{M}_{tm}(\oplus)$. It is rather obvious that $H$ can be obtained in logarithmic space from $S$, which establishes a logarithmic-space reduction from $\mathrm{M}_{tm}(\oplus)$ to xNoMW(2).

■

**Corollary 6.9** $\mathrm{M}_{tm}(\oplus)$ *is in* P *and* $\mathrm{C_{=}L}$*-hard.*

**Proof:** Due to Corollary 4.15, xNoMW(2) is contained in P. Since P is closed under $\leq_m^{\mathrm{L}}$-reducibility, $\mathrm{M}_{tm}(\oplus)$ is contained in P due to Lemma 6.8. The $\mathrm{C_{=}L}$ lower complexity bound results from reducing xPOM(2) to $\mathrm{M}_{tm}(\oplus)$ and by Corollary 4.15.

■

The result of Corollary 6.9 can be obtained by a much easier construction than the one that is based on Lemma 6.8. This easier algorithm, however, would only prove an upper complexity bound without emphasising the fact that the four problems xNoW(2), xPOM(2), xNoMW(2) and $\mathrm{M}_{tm}(\oplus)$ can be considered equally complicated in some sense. This inexactness is mainly caused by the relation between the problems xNoMW(2) and xNoW(2) established by an $\leq_m^{\mathrm{NL}}$-reduction (Theorem 4.4).

Since the second mentioned way to show polynomial-time decidability of $\mathrm{M}_{tm}(\oplus)$ is interesting in its own right but also of importance for following constructions, we discuss its main part. Let $S = (\mathcal{F}, A)$ be a recurrent $\{\oplus\}$-system of dimension $n$

where $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ and $A = (a_1, \ldots, a_n)$. Let $\mathbf{x} =_{\text{def}} (x_1, \ldots, x_n)$. For each $i \in \{1, \ldots, n\}$, it holds that $f_i(\mathbf{x}) = \sum_{j=1}^{n} c_j^i \cdot x_j$ for $c_j^i \in \mathbb{N}$, $j \in \{1, \ldots, n\}$. Since $f_i$ is given as an arithmetical circuit $C_i$, $c_j^i$ is equal to the number of paths in $C_i$ from the input vertex with label $j$ to the output vertex. We define an $n \times n$ matrix $M$ as: $M_{ij} =_{\text{def}} c_j^i$ for all $i, j \in \{1, \ldots, n\}$. We will call $M$ the *function matrix* of $S$. It holds for every $t \geq 0$ that $S(t) = (M^t A)_n$. If we want to obtain an instance of xPOM(2), we define another matrix $N$ as

$$N =_{\text{def}} \begin{pmatrix} c_1^1 & c_2^1 & \ldots & c_n^1 & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_1^n & c_2^n & \ldots & c_n^n & a_n \\ 0 & 0 & \ldots & 0 & 0 \end{pmatrix}.$$

It obviously holds $(M^k A)_n = (N^{k+1})_{n,n+1}$ for $k \geq 0$. It is not difficult to see that matrix $N$ can be computed in polynomial time from $S$. This construction is illustrated by our preferred example, the Fibonacci numbers. Let $\text{Fib} = (\mathcal{F}_{\text{Fib}}, A_{\text{Fib}})$ be a recurrent $\{\oplus\}$-system whose components are defined as follows:

$$\mathcal{F}_{\text{Fib}} =_{\text{def}} \langle f_1(x_1, x_2) =_{\text{def}} x_2, f_2(x_1, x_2) =_{\text{def}} x_1 \oplus x_2 \rangle$$
$$A_{\text{Fib}} =_{\text{def}} (0, 1).$$

We already encountered this example in Chapter 5. The constructions of matrices $M$ and $N$ above yield the following:

$$M =_{\text{def}} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad N =_{\text{def}} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

For the first few powers of $N$, we obtain these results:

$$N^1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$N^2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$N^3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

$$N^4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 2 \\ 3 & 5 & 3 \\ 0 & 0 & 0 \end{pmatrix}.$$

Remember that, by construction, the output or result of each iteration step is the 2, 3-entry of the corresponding power of $N$.

The major goal of this section is to determine the complexity of $\mathrm{M}_{tm}(\cap, \oplus)$. Corollary 6.7 shows a possible way. As we have explained, the solution will consist of two parts, and the first part has now been concluded by the results of Corollary 6.9. The second part mainly consists of a solution of the question, given a recurrent $\{\cap,\oplus\}$-system $S$ and a number $t$, whether $S(t)=\emptyset$. This problem is captured by the following definition.

Problem definition

---

EMPTY($\mathcal{O}$) for $\mathcal{O}\subseteq\{\cup,\cap,\overline{\phantom{x}},\oplus,\otimes\}$ (emptiness problem for recurrent $\mathcal{O}$-systems).
**INSTANCE** $(S,t)$ where $S$ is a recurrent $\mathcal{O}$-system and $t\geq 0$, and $t$ is given in binary representation.
**QUESTION** $S(t)=\emptyset$?

---

It is clear that the emptiness problem becomes difficult only for those recurrent systems that contain intersection or complementation operations, since recurrent $\{\cup,\oplus,\otimes\}$-systems compute only non-empty sets. Here, we are interested in the complexity of EMPTY$(\cap,\oplus)$. Unfortunately, we cannot give an efficient decision algorithm for EMPTY$(\cap,\oplus)$. However, for deciding $\mathrm{M}_{tm}(\cap,\oplus)$ in polynomial time it suffices to consider only a subset of EMPTY$(\cap,\oplus)$. This subset is selected by a structural property of recurrent systems.

Let $G=(V,A)$ be a directed graph. A cycle of length $k$ in $G$ is a sequence $C=(x_1,\ldots,x_k)$ of vertices of $G$ such that $C$ is an $x_1,x_k$-path in $G$ and $(x_k,x_1)\in A$. Hence, cycles contain at least one vertex. Let $S=(\mathcal{F},A)$ be a recurrent $\{\cap,\oplus\}$-system of dimension $n$ where $\mathcal{F}=\langle f_1,\ldots,f_n\rangle$. We obtain $S'$ from $S$ by replacing every $\cap$ in the functions of $S$ by $\oplus$; $S'$ is a recurrent $\{\oplus\}$-system. Let $M'$ be the function matrix of $S'$, as it is defined above. A non-zero entry in $M'$ means existence of a path from input to output vertex in a circuit representation. We can interpret $M'$ as the adjacency matrix of a directed graph. Let $G_S=(\{1,\ldots,n\},A_S)$ be the graph on vertices $1,\ldots,n$ where $(i,j)\in A_S$ if and only if $M'_{ij}\neq 0$. Equivalently, $(i,j)\in A_S$ if and only if $f_i$ depends on its $j$-th input component. Function $f_i$ *appears in a cycle* of length $k$, if there is a cycle of length $k$ in $G_S$ containing vertex $i$. The problem $\circ$-EMPTY$(\cap,\oplus)$ is the set of pairs $(S,t)$ in EMPTY$(\cap,\oplus)$ where the output function of $S$ appears in a cycle.

First, we show that $\circ$-EMPTY$(\cap,\oplus)$ can be decided in polynomial time, which is heavily based on a lemma of linear algebra. Let $L$ be a $1\times n$ matrix over $\mathbb{Z}$, $n\geq 1$. Let $[L]^{=0}=_{\mathrm{def}}\{\mathbf{x}\in\mathbb{R}^n : L\mathbf{x}=0\}$ denote the hyperplane of $\mathbb{R}^n$ defined by $L$.

**Lemma 6.10** *Let $M$ be an $n\times n$ matrix over $\mathbb{N}$, and let $L$ be a $1\times n$ matrix over $\mathbb{Z}$. Let $\mathbf{a}\in\mathbb{N}^n$. If $M^i\mathbf{a}\in[L]^{=0}$ for all $i\in\{0,\ldots,n-1\}$ then $M^i\mathbf{a}\in[L]^{=0}$ for all $i\geq 0$.*

**Proof:** Let $k\leq n-1$ be smallest possible such that $M^{k+1}\mathbf{a}=\sum_{i=0}^{k}c_i\cdot M^i\mathbf{a}$ for some $c_0,\ldots,c_k\in\mathbb{R}$. By induction, it holds that, for every $m\geq k+1$, there are $x_0,\ldots,x_k\in\mathbb{R}$ such that $M^m\mathbf{a}=\sum_{i=0}^{k}x_i\cdot M^i\mathbf{a}$: Let $x_0,\ldots,x_k\in\mathbb{R}$ such that

$M^r\mathbf{a} = \sum_{i=0}^{k} x_i \cdot M^i\mathbf{a}$ for $r \geq k+1$. Then,

$$M^{r+1}\mathbf{a} = \sum_{i=1}^{k}(x_{i-1} + x_k c_i) \cdot M^i\mathbf{a} + x_k c_0 \cdot M^0\mathbf{a}\,.$$

By assumption, $M^i\mathbf{a} \in [L]^{=0}$, $i \leq k$. Then, $M^i\mathbf{a} \in [L]^{=0}$ for every $i \geq 0$.

■

**Lemma 6.11** $\circ$-EMPTY$(\cap, \oplus)$ *is in* P.

**Proof:** Let $(S,t)$ be an instance of $\circ$-EMPTY$(\cap,\oplus)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\cap,\oplus\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1,\ldots,f_n\rangle$. We apply Lemma 6.6 and obtain functions $f'_1,\ldots,f'_n$ and $g^1_1,\ldots,g^1_{r_1},g^2_1,\ldots,g^n_{r_n}$ such that, for every $i \in \{1,\ldots,n\}$ and every $\mathbf{a} \in \mathbb{N}^n$,

$$\begin{aligned} g^i_j(\mathbf{a}) = 0 \text{ for all } j \in \{1,\ldots,r_i\} \quad &\Longrightarrow \quad f_i(\mathbf{a}) = f'_i(\mathbf{a}) \\ g^i_j(\mathbf{a}) \neq 0 \text{ for some } j \in \{1,\ldots,r_i\} \quad &\Longrightarrow \quad f_i(\mathbf{a}) = \emptyset\,. \end{aligned}$$

We observe the following important and easy property: for every $i \in \{1,\ldots,n\}$ and $t \in \mathbb{N}$, $S[f_i](t+1) = \emptyset$ if and only if $f'_i(\mathcal{F}(t)) = \emptyset$ or there is $j \in \{1,\ldots,r_i\}$ such that $g^i_j(\mathcal{F}(t)) \neq 0$. Let $k \leq n$ be the smallest number such that $f_n$ appears in a cycle of length $k$. If there is $t' \leq 2n^2$ such that $S(t') = \emptyset$ and $t' \equiv t \pmod{k}$, then $S(t) = \emptyset$. We will show that this implication can be strengthened to an equivalence.

Let $t > 2n^2$, and let $S(t) = \emptyset$. Let $W =_{\text{def}} (\nu_0,\nu_1,\ldots,\nu_r)$ be a longest walk in $G_S$ such that $\nu_0 = n$ and $S[f_{\nu_i}](t-i) = \emptyset$, $i \in \{0,\ldots,r\}$. Note that $r < t$ and $f_{\nu_r}$ does not depend on a variable $x_i$ such that $S[f_i](t-r-1) = \emptyset$. We might say that $S[f_{\nu_r}](t-r) = \emptyset$ *causes* $S(t) = \emptyset$, which is proven by walk $W$. Consider the subsequence $\omega =_{\text{def}} (\nu_0,\nu_k,\ldots,\nu_{sk})$ of $W$ where $s \cdot k \leq r < (s+1)\cdot k$. If a number appears twice in $\omega$, say $\nu_j$ and $\nu_{j'}$, $j < j'$, then $W' =_{\text{def}} (\nu_0,\nu_1,\ldots,\nu_j,\nu_{j'+1},\ldots,\nu_r)$ proves that $S(t-(j'-j)) = S[f_n](t-(j'-j)) = \emptyset$. Note that $t \equiv t-(j'-j) \pmod{k}$. We apply this argument repeatedly and as often as possible, i.e., we reduce the number of elements in $W'$ to at most $(n-1)k+1$, and obtain that, if $S[f_{\nu_r}](t') = \emptyset$ for $t' \in \mathbb{N}$, then $S(t'') = \emptyset$ for $t'' \geq t'$ where $t'' \equiv t'+r \pmod{k}$ and $t'+kn-k < t'' \leq t'+kn \leq t'+n^2$. Let $t_0 \in \{1,\ldots,k\}$ such that $t_0 \equiv t-r \pmod{k}$. Then, for all $i \geq 0$ such that $t_0 + ik < t-r$, it holds that $S[f_{\nu_r}](t_0+ik) \neq \emptyset$ by the maximality of $W$.

Let $M$ be the function matrix belonging to the function sequence $\langle f'_1,\ldots,f'_n\rangle$. Let $T =_{\text{def}} M^k$ and $A_{t_0} =_{\text{def}} M^{t_0-1}A$. Due to Lemma 6.6, if $S[f_{\nu_r}](t_0) \neq \emptyset$ then $S[f_{\nu_r}](t_0) = f'_{\nu_r}(A_{t_0})$. For $i < n$, if $S[f_{\nu_r}](t_0+ik) \neq \emptyset$ then $S[f_{\nu_r}](t_0+ik) = f'_{\nu_r}(T^iA_{t_0})$. Furthermore, $S[f_{\nu_r}](t_0+ik) \neq \emptyset$ if and only if $g^{\nu_r}_j(T^iA_{t_0}) = 0$ for all $j \in \{1,\ldots,r_{\nu_r}\}$. We apply Lemma 6.10 and obtain that $S[f_{\nu_r}](t_0+(n-1)\cdot k) \neq \emptyset$ if and only if $g^{\nu_r}_j(T^iA_{t_0}) = 0$ for all $i \in \{0,\ldots,n-1\}$ and $j \in \{1,\ldots,r_{\nu_r}\}$. Since $S[f_{\nu_r}](t-r) = \emptyset$, there is $i < n$ such that $S[f_{\nu_r}](t_0+ik) = \emptyset$, where $t_0+ik \leq k+(n-1)\cdot k \leq n^2$. Hence, if $S(t) = \emptyset$ then there is $t' \leq 2n^2$ such that $S(t') = \emptyset$

and $t \equiv t' \pmod{k}$. It takes polynomial time to compute $\mathcal{F}(0), \ldots, \mathcal{F}(2n^2)$, which concludes the proof.

■

Before we finally consider $\mathrm{M}_{tm}(\cap, \oplus)$, we want to illustrate the construction of the proof of the following theorem by discussing an example. Consider Figure 10. It is shown graph $G_S$ of a 9-dimensional recurrent $\{\cap, \oplus\}$-system $S$. For instance, function $f_1$ depends only on $x_1$, whereas function $f_2$ depends on $x_1$ and $x_2$. Lemma 6.11 shows that we can decide for every $t \geq 0$ in polynomial time whether $S[f_2](t) = \emptyset$ or whether $S[f_5](t) = \emptyset$. We want to decide whether $S[f_9](20) = 8$. Then, it must hold that $S[f_2](18) \neq \emptyset$, $S[f_5](18) \neq \emptyset$, $S[f_5](19) \neq \emptyset$ and $S[f_6](18) \neq \emptyset$. Furthermore, $S[f_2](18)$, $S[f_5](18)$, $S[f_5](19)$ and $S[f_6](18)$ must not all have values greater than 8. Applying the method of Corollary 6.7, we obtain a recurrent $\{\oplus\}$-system $S'$ that is equivalent to $S$ in iteration steps that do not produce the empty set. Using the function matrix of $S'$, we can compute these values in polynomial time or decide that one of them exceeds 8 (remember the discussion following the proof of Lemma 4.13). Finally, we can straightforwardly evaluate $S[f_7](19)$, $S[f_8](19)$ and $S[f_9](20)$ in polynomial time.

**Theorem 6.12** $\mathrm{M}_{tm}(\cap, \oplus)$ *is in* P.

**Proof:** Let $(S, t, b)$ be an instance of $\mathrm{M}_{tm}(\cap, \oplus)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\cap, \oplus\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$. Let $S' = (\mathcal{F}', A)$ be the recurrent $\{\oplus\}$-system that is obtained from $S$ by application of Corollary 6.7. It holds that if $S(t) \neq \emptyset$ then $S'(t) = S(t)$. If the output function $f_n$ of $S$ is contained in a cycle then $(S, t, b) \in \mathrm{M}_{tm}(\cap, \oplus)$ if and only if $(S', t, b) \in \mathrm{M}_{tm}(\oplus)$ and $(S, t) \notin \circ\text{-}\mathrm{EMPTY}(\cap, \oplus)$, which can be verified in polynomial time due to Corollary 6.9 and Lemma 6.11.

So, let $f_n$ be not contained in a cycle. If $t < n$, $S(t)$ can be computed straightforwardly. Let $t \geq n$. Consider graph $G_S$. Since vertex $n$ does not appear in a cycle in $G_S$ there is a maximal connected subgraph $G^\diamondsuit$ of $G_S$ that contains vertex $n$ and that does not contain a vertex that appears in a cycle. (In Figure 10 this subgraph is induced by the vertex set $\{7, 8, 9\}$.) It holds that every vertex in $G^\diamondsuit$ has a predecessor in $G_S$, and all those predecessors that do not occur in $G^\diamondsuit$ are contained in cycles in $G_S$. Let $\mathfrak{F}$ be the set of predecessors of vertices in $G^\diamondsuit$ that are not contained in $G^\diamondsuit$. (In Figure 10 this is the set $\{2, 5, 6\}$.) $G^\diamondsuit$ is acyclic, and therefore every walk in $G^\diamondsuit$ contains less than $n$ vertices. If we know all values $S[f_i](t')$ where $i \in \mathfrak{F}$ and $t - n < t' \leq t$ we can compute $S[f_n](t)$. However, for large $t$ these values can become large. It suffices though to consider only numbers that are not larger than $b$, since $\oplus$ is monotone (it does not matter whether $S[f_n](t) = \emptyset$ or $S[f_n](t) > b$, if $S[f_n](t) \neq b$). Using the function matrix for $S'$, we can compute these values in polynomial time for all functions with index in $\mathfrak{F}$ (remember the discussion in the preceding example). If $S[f_i](t') > b$ we can also assume $S[f_i](t') = \emptyset$ without causing any error for our problem. In polynomial time, $\mathcal{F}(t-n+1), \ldots, \mathcal{F}(t)$ can be computed (where numbers greater than $b$ are replaced by $\emptyset$). Finally, accept if and only if the last component

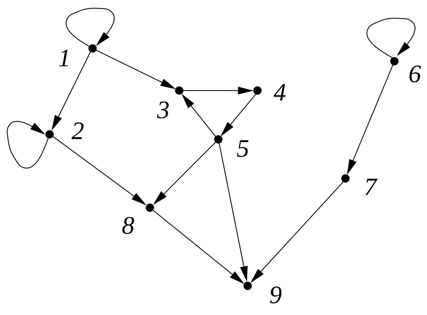


**Figure 10** Graph $G_S$ for some sample recurrent system $S$.

of $\mathcal{F}(t)$ is equal to $b$, i.e., if and only if $S(t) = b$.

It remains to consider the problems of generating $G_S$ and $\mathfrak{F}$. If a vertex in a directed graph is contained in a cycle, there is a cycle of length at most the number of vertices of the graph. Hence, determining whether a vertex appears in a cycle can be done in nondeterministic logarithmic space. Starting with vertex $n$ in $G_S$, $G^\diamondsuit$ can be generated straightforwardly. Set $\mathfrak{F}$ is also computed easily by determining the predecessors of every vertex in $G^\diamondsuit$ and choosing those that do not appear in a cycle. All this can be done in polynomial time.

■

**Theorem 6.13** $\mathrm{M}_{ex}(\cap)$, $\mathrm{M}_{ex}(\oplus)$, $\mathrm{M}_{ex}(\otimes)$ *and* $\mathrm{M}_{ex}(\cap, \oplus)$ *are* NP*-complete.*

**Proof:** For hardness of the problems we first show that SORSR reduces to $\mathrm{M}_{ex}(\cap)$. Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ be an instance of SORSR; let $R_1, \ldots, R_k$ be registers of sizes $b_1, \ldots, b_k$, respectively. We define a recurrent $\{\cap\}$-system $S = (\mathcal{F}, A)$ as follows. For every $i \in \{1, \ldots, k\}$, for every $j \in \{2, \ldots, b_i\}$, let

$$f^{(i)}_{j-1}(\mathbf{x}) =_{\mathrm{def}} x^{(i)}_j \quad \text{and} \quad f^{(i)}_{b_i}(\mathbf{x}) =_{\mathrm{def}} x^{(i)}_1$$

where $\mathbf{x} =_{\mathrm{def}} (x^{(1)}_1, \ldots, x^{(1)}_{b_1}, x^{(2)}_1, \ldots, x^{(k)}_{b_k}, x')$. Let $A =_{\mathrm{def}} (c^{(1)}_1, \ldots, c^{(k)}_{b_k}, 0)$ where $c^{(i)}_j \in \{0,1\}$ and $c^{(i)}_j = 1$ if and only if $R_i(j) = 1$. Furthermore, let $f'(\mathbf{x}) =_{\mathrm{def}} x^{(1)}_1 \cap \cdots \cap x^{(k)}_1$ and $\mathcal{F} =_{\mathrm{def}} \langle f^{(1)}_1, \ldots, f^{(k)}_{b_k}, f' \rangle$. Informally spoken, when we regard graph $G_S$, every register of $\mathcal{R}$ corresponds to a cycle in $G_S$ of length the size of the register. It holds that $S[f^{(i)}_j](t) = 1$ if and only if $R_i(j) = 1$ after application of $t$ left ring-shift operations to $R_i$, and $S[f'](t+1) = 1$ if and only if $S[f^{(1)}_1](t) = \cdots = S[f^{(k)}_1](t) = 1$. Hence, $(S, 1) \in \mathrm{M}_{ex}(\cap)$ if and only if $\mathcal{R} \in$ SORSR. So, by Theorem 4.20, $\mathrm{M}_{ex}(\cap)$ is NP-hard.

We now reduce $\mathrm{M}_{ex}(\cap)$ to $\mathrm{M}_{ex}(\oplus)$ and $\mathrm{M}_{ex}(\otimes)$. Let $(S, b)$ be an instance of $\mathrm{M}_{ex}(\cap)$ where $S = (\mathcal{F}, A)$. We define recurrent $\{\oplus\}$- and $\{\otimes\}$-systems $S'$ and $S''$ as follows. Let $\mathcal{F}'$ and $\mathcal{F}''$ emerge from $\mathcal{F}$ by replacing each occurrence of $\cap$ by $\oplus$ and $\otimes$, respectively, i.e., we only change vertex labels. Let $A'$ emerge from $A$ by replacing $b$ by 0 and all other numbers by 1. Similarly, obtain $A''$ from $A$ by replacing $b$ by 1 and all other numbers by 0. Let $S' =_{\mathrm{def}} (\mathcal{F}', A')$ and $S'' =_{\mathrm{def}} (\mathcal{F}'', A'')$. Then, $(S, b) \in \mathrm{M}_{ex}(\cap)$ if and only if $(S', 0) \in \mathrm{M}_{ex}(\oplus)$ if and only if $(S'', 1) \in \mathrm{M}_{ex}(\otimes)$. Finally, $\mathrm{M}_{ex}(\oplus)$ reduces to $\mathrm{M}_{ex}(\cap, \oplus)$ due to Lemma 5.2. Hence, $\mathrm{M}_{ex}(\oplus)$, $\mathrm{M}_{ex}(\otimes)$ and $\mathrm{M}_{ex}(\cap, \oplus)$ are NP-hard.

For containment, we show that $\mathrm{M}_{ex}(\cap, \oplus)$ and $\mathrm{M}_{ex}(\otimes)$ are contained in NP. In both cases, we will find a sufficiently small number $f(S, b)$ such that, for $S$ a recurrent $\{\cap, \oplus\}$- or $\{\otimes\}$-system, $b \in [S]$ if and only if there is $t < f(S, b)$ and $b \in S(t)$. First, we need a technical definition. Let $f = f(x_1, \ldots, x_k)$ be a $\{\cap, \oplus\}$-function, and let $b, b' \in \mathbb{N}$, $b' \leq b$. Let $A_1, \ldots, A_k \subseteq \mathbb{N}$ be sets of cardinality at most 1. For every $i \in \{1, \ldots, k\}$, it holds that, if $A_i \subseteq \mathbb{N} \oplus \{b+1\}$, then $f(A_1, \ldots, A_k) = \{b'\}$ if and only if $f(A_1, \ldots, A_{i-1}, \emptyset, A_{i+1}, \ldots, A_k) = \{b'\}$. In this sense we say that $(A_1, \ldots, A_k)$ and

$(A_1, \ldots, A_{i-1}, \emptyset, A_{i+1}, \ldots, A_k)$ are *similar with respect to $b$*; we extend the definition of similar tuples to the transitive closure. Now, let $S = (\mathcal{F}, A)$ be a recurrent $\{\cap, \oplus\}$-system of dimension $n$. Let $t_1, t_2 \geq 0$, $t_1 < t_2$, such that $\mathcal{F}(t_1)$ and $\mathcal{F}(t_2)$ are similar with respect to $b$. Then, $\mathcal{F}(t_1+1)$ and $\mathcal{F}(t_2+1)$ are also similar with respect to $b$. For an upper bound on the number of iteration steps that are necessary to generate $b'$ by $S$, if possible, it suffices to determine the maximal number of iteration steps before $S$ generates a tuple $\mathcal{F}(t)$ that is similar with respect to $b$ to an already generated tuple. Each tuple has $n$ components, and each component can contain $b+2$ different values: $\{0\}, \ldots, \{b\}$ and $\emptyset$ (which is similar with respect to $b$ to $\{b+1\}, \{b+2\}, \ldots$). This number sums up to $(b+2)^n \leq 2^{1+n\log b}$, which is of polynomial size in the length of $S$ and $b$. Since $\mathrm{M}_{tm}(\cap, \oplus) \in \mathrm{P}$, a nondeterministic polynomial-time algorithm can decide whether $(S, b) \in \mathrm{M}_{ex}(\cap, \oplus)$ by checking $(S, t, b) \in \mathrm{M}_{tm}(\cap, \oplus)$ for every $t \leq 2^{1+n\log b}$. Hence, $\mathrm{M}_{ex}(\cap)$, $\mathrm{M}_{ex}(\oplus)$ and $\mathrm{M}_{ex}(\cap, \oplus)$ are NP-complete.

It remains to find a complexity upper bound for $\mathrm{M}_{ex}(\otimes)$. Let $(S, b)$ be an instance of $\mathrm{M}_{ex}(\otimes)$. We consider two cases. If $b = 0$, we can use the algorithm for $\mathrm{M}_{ex}(\cup)$, since instances of $\mathrm{M}_{ex}(\otimes)$ of the form $(S', 0)$ easily reduce to instances of $\mathrm{M}_{ex}(\cup)$: simply replace every $\otimes$ in the functions of $S$ by $\cup$ and obtain $S''$. It holds that $S(t) = 0$ if and only if $0 \in S''(t)$ for every $t \geq 0$, which can be shown by an easy induction. So, let $b \geq 1$. Then, however, we can proceed as above. We define an analogue notion of similarity, where additionally 0 is to be treated like numbers greater than $b$. The number of the corresponding equivalence classes for $n$-tuples is bounded above by $(b+1)^n$. So, $\mathrm{M}_{ex}(\otimes)$ is in NP due to Lemma 6.3 and hence NP-complete.

■

## 6.3 Polynomial space decidable membership problems

Besides the announced PSPACE-completeness results, we will show three theorems that are interesting in their own right. First, we will show that the satisfiability problem for quantified Boolean formulas, denoted as QBF, reduces to $\mathrm{M}_{ex}(\cup, \cap)$. Together with the PSPACE-completeness of QBF, we can conclude PSPACE-hardness of $\mathrm{M}_{ex}(\cup, \cap)$ and $\mathrm{M}_{ex}(\cup, \cap, ^{-})$. Second, we will show that the exact membership problem for recurrent $\{\cup, \oplus, \otimes\}$-systems can be decided in polynomial space. This is surprising when we remember that $\mathrm{MC}(\cup, \oplus, \otimes)$, the membership problem for arithmetical $\{\cup, \oplus, \otimes\}$-circuits, is PSPACE-complete [87] (see also Theorem 5.7). Third, we will show that a recurrent $\{\cup, \oplus, \otimes\}$-system $S$ of dimension $n$ needs at most $(b+1) \cdot 2^{n^3}$ iteration steps to generate number $b$, if it is possible. In the end, this will lead to a polynomial-space decision algorithm for $\mathrm{M}_{ex}(\cup, \oplus, \otimes)$.

For our first goal, consider the following definition. A *quantified Boolean formula* is a first-order logic formula without functions, predicates and free variables. We assume that every quantified Boolean formula $H$ is of the form

$$H = Q_1 x_1 \ldots Q_n x_n H'(x_1, \ldots, x_n)$$

where $Q_1, \dots, Q_n \in \{\exists, \forall\}$ and $H'(x_1, \dots, x_n)$ is a Boolean formula with $\wedge$, $\vee$ and $\neg$ and every $\neg$ applies only to variables $x_i$, $i \in \{1, \dots, n\}$. Every quantified Boolean formula evaluates to *true* or *false* (1 or 0, respectively). The problem QBF is the set of true quantified Boolean formulas. This problem was introduced by Stockmeyer and Meyer [79].

For a reduction from QBF to $\mathrm{M}_{ex}(\cup, \cap)$, we need to enumerate all binary $n$-tuples over $\{\text{false}, \text{true}\}$ iteratively. Since $\{\cup, \cap\}$-functions compute sets, that may contain more than one single element, the Boolean values are represented by answers to appropriate containment questions. Consider the following $\{\cup, \cap\}$-functions:

$$\zeta_1(a, \overline{a}, e, \overline{e}, c, \overline{c}) =_{\text{def}} (((a \cap \overline{c}) \cup (\overline{a} \cap c)) \cap \overline{e}) \cup (e \cap c) \quad \text{and}$$
$$\zeta_2(a, \overline{a}, e, \overline{e}, c, \overline{c}) =_{\text{def}} (((a \cap c) \cup (\overline{a} \cap \overline{c})) \cap \overline{e}) \cup (e \cap \overline{c})\,.$$

The following lemma is easy to prove. The operator $\triangle$ denotes the symmetric difference, i.e., the set-theoretic analogue of the Boolean xor-function.

**Lemma 6.14** *Let $A, B, C, D, E, F \subseteq \mathbb{N}$ and $b \in \mathbb{N}$.*
*If $b \in (A \triangle B) \cap (C \triangle D) \cap (E \triangle F)$ then*

*(1) $b \in \zeta_1(A, B, C, D, E, F) \ \triangle\ \zeta_2(A, B, C, D, E, F)$*

*(2) $b \in \zeta_1(A, B, C, D, E, F) \ \triangle\ E$ if and only if $b \in A \cap D$.*

**Proof:** The claim can be proved by simply checking all eight input situations. We give a table containing all situations of a number $b$ with respect to the input sets. The symbol $*$ means that given number $b$ is contained in that set.

| $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $\zeta_1(A,B,C,D,E,F)$ | $\zeta_2(A,B,C,D,E,F)$ |
|---|---|---|---|---|---|---|---|
|  | $*$ |  | $*$ |  | $*$ |  | $*$ |
|  | $*$ |  | $*$ | $*$ |  | $*$ |  |
|  | $*$ | $*$ |  |  | $*$ |  | $*$ |
|  | $*$ | $*$ |  | $*$ |  | $*$ |  |
| $*$ |  |  | $*$ |  | $*$ | $*$ |  |
| $*$ |  |  | $*$ | $*$ |  |  | $*$ |
| $*$ |  | $*$ |  |  | $*$ |  | $*$ |
| $*$ |  | $*$ |  | $*$ |  | $*$ |  |

Statement (1) is obvious by comparing the last two columns of the table. Statement (2) is similarly easy to observe.

■

Using functions $\zeta_1$ and $\zeta_2$ we construct a recurrent $\{\cup, \cap\}$-system that enumerates all binary $n$-tuples for $n \geq 1$. Let $\mathbf{u} =_{\text{def}} (u_1, \overline{u}_1, \dots, u_n, \overline{u}_n)$. We define:

$$f_1(\mathbf{u}) =_{\text{def}} \overline{u}_1$$
$$f'_1(\mathbf{u}) =_{\text{def}} u_1$$
$$f_{i+1}(\mathbf{u}) =_{\text{def}} \zeta_1(u_i, \overline{u}_i, f_i(\mathbf{u}), f'_i(\mathbf{u}), u_{i+1}, \overline{u}_{i+1})$$
$$f'_{i+1}(\mathbf{u}) =_{\text{def}} \zeta_2(u_i, \overline{u}_i, f_i(\mathbf{u}), f'_i(\mathbf{u}), u_{i+1}, \overline{u}_{i+1})\,,\ i \in \{1, \dots, n-1\}\,.$$

Every function can be represented by a $\{\cup, \cap\}$-circuit using at most $c \cdot n$ vertices for some constant $c$. (A close look reveals $c < 20$.) Let

$$\begin{aligned}
\mathcal{F}^n_{\text{enum}} &=_{\text{def}} \langle f_1, f_1', \ldots, f_n, f_n' \rangle \\
A^n_{\text{enum}} &=_{\text{def}} (\{0\}, \{1\}, \ldots, \{0\}, \{1\}),
\end{aligned}$$

and let $S^n_{\text{enum}} =_{\text{def}} (\mathcal{F}^n_{\text{enum}}, A^n_{\text{enum}})$. The following lemma shows that $\mathcal{F}^n_{\text{enum}}(t)$ in reversed order can be interpreted as a representation of the $n$ least significant bits of the binary representation of $t \in \mathbb{N}$. Let $\text{bin}_i(t)$, $i \geq 0$, denote the $i$-th least significant bit of the binary representation of $t$; the count starts with 0, i.e., $\text{bin}_i(t)$ is the coefficient of $2^i$ in the binary representation of $t$.

**Lemma 6.15** *For every $t \in \mathbb{N}$ and $i \in \{1, \ldots, n\}$:*

*(1)* $\{0, 1\} = S^n_{\text{enum}}[f_i](t) \ \triangle \ S^n_{\text{enum}}[f_i'](t)$

*(2)* $1 \in S^n_{\text{enum}}[f_i](t) \iff \text{bin}_{i-1}(t) = 1$.

**Proof:** We prove the statements of the lemma each by induction over $t$. For statement (1), observe that the claim is true for $t = 0$ and all $i \in \{1, \ldots, n\}$ and for $i = 1$ and all $t \geq 0$. Now, let the statement be true for $t \geq 0$, and assume that it holds for $i < n$. We want to prove that it holds for $t + 1$ and $i + 1$. By induction hypothesis,

$$\begin{aligned}
\{0, 1\} &= S^n_{\text{enum}}[f_i](t) \quad && \triangle \ S^n_{\text{enum}}[f_i'](t) \\
&= S^n_{\text{enum}}[f_i](t+1) && \triangle \ S^n_{\text{enum}}[f_i'](t+1) \\
&= S^n_{\text{enum}}[f_{i+1}](t) && \triangle \ S^n_{\text{enum}}[f_{i+1}'](t).
\end{aligned}$$

Hence, the prerequisites of Lemma 6.14 are fulfilled using 0 and 1. And we conclude that

$$\{0, 1\} = S^n_{\text{enum}}[f_{i+1}](t+1) \ \triangle \ S^n_{\text{enum}}[f_{i+1}'](t+1),$$

which proves statement (1).

For statement (2), observe that the claim is true for $i = 1$. Let $i < n$. First, note that $S^n_{\text{enum}}[f_{i+1}](0) = \{0\}$. Consider $S^n_{\text{enum}}[f_{i+1}](t+1)$. If $\text{bin}_{i-1}(t) \leq \text{bin}_{i-1}(t+1)$ then $\text{bin}_i(t) = \text{bin}_i(t+1)$. By induction hypothesis, it holds that

$$\begin{aligned}
&1 \notin S^n_{\text{enum}}[f_i](t) \ \text{ and } \ 1 \notin S^n_{\text{enum}}[f_i](t+1) \quad \text{or} \\
&1 \notin S^n_{\text{enum}}[f_i](t) \ \text{ and } \ 1 \in S^n_{\text{enum}}[f_i](t+1) \quad \text{or} \\
&1 \in S^n_{\text{enum}}[f_i](t) \ \text{ and } \ 1 \in S^n_{\text{enum}}[f_i](t+1).
\end{aligned}$$

Applying Lemma 6.14, which means $1 \notin A \cap D$ in statement (2), we obtain that either $1 \in S^n_{\text{enum}}[f_{i+1}](t)$ and $1 \in S^n_{\text{enum}}[f_{i+1}](t+1)$ or $1 \notin S^n_{\text{enum}}[f_{i+1}](t)$ and $1 \notin S^n_{\text{enum}}[f_{i+1}](t+1)$, which concludes this case. If $\text{bin}_{i-1}(t) > \text{bin}_{i-1}(t+1)$ then $\text{bin}_i(t) \neq \text{bin}_i(t+1)$. By induction hypothesis,

$$1 \in S^n_{\text{enum}}[f_i](t) \ \text{ and } \ 1 \notin S^n_{\text{enum}}[f_i](t+1),$$

and applying Lemma 6.14, where $1 \in A \cap D$ holds, we obtain

$$1 \in S^n_{\text{enum}}[f_{i+1}](t) \;\triangle\; S^n_{\text{enum}}[f_{i+1}](t+1)\,,$$

which shows the claim.

■

Let $H = Q_1x_1 \ldots Q_nx_nH'(x_1, \ldots, x_n)$ be a quantified Boolean formula. For $t \in \mathbb{N}$ and $i \in \{0, \ldots, n\}$, let

$$H^{(i)}(t) =_{\text{def}} Q_{n-i+1}x_{n-i+1} \cdots Q_nx_nH'(\text{bin}_{n-1}(t), \ldots, \text{bin}_i(t), x_{n-i+1}, \ldots, x_n)\,.$$

Note that the functions $\text{bin}_i$ are ordered by decreasing index whereas the variables $x_j$ are ordered by increasing index. Two particularly special cases are $H^{(n)}(t)$ and $H^{(0)}(t)$:

$$\begin{aligned} H^{(n)}(t) &= Q_1x_1 \ldots Q_nx_nH'(x_1, \ldots, x_n) \\ H^{(0)}(t) &= H'(\text{bin}_{n-1}(t), \ldots, \text{bin}_0(t))\,. \end{aligned}$$

By definition, the following equivalences hold:

$$\begin{aligned} Q_1 = \exists &\quad\Longrightarrow\quad H \equiv (H^{(n-1)}(0) \vee H^{(n-1)}(2^{n-1})) \\ Q_1 = \forall &\quad\Longrightarrow\quad H \equiv (H^{(n-1)}(0) \wedge H^{(n-1)}(2^{n-1}))\,. \end{aligned}$$

This evaluation procedure inductively defines an evaluation tree for $H$, where the values of $H^{(n-1)}(0)$ and $H^{(n-1)}(2^{n-1})$ are the values of the predecessors of a final $\vee$- or $\wedge$-vertex, depending on the nature of $Q_1$. Then, $H^{(n)}(0)$ denotes the value of this final vertex. Hence, $H^{(i)}(t)$ for $i \in \{0, \ldots, n\}$ and $t \geq 0$ represents the value of some $\vee$- or $\wedge$-vertex in that tree.

**Theorem 6.16** QBF $\leq^{\text{L}}_m$ $\text{M}_{ex}(\cup, \cap)$.

**Proof:** Let $H = Q_1x_1 \ldots Q_nx_nH'(x_1, \ldots, x_n)$ be a quantified Boolean formula in the variables $x_1, \ldots, x_n$. We will define a recurrent $\{\cup, \cap\}$-system that evaluates the evaluation tree of $H$. Let $S =_{\text{def}} (\mathcal{F}, A)$ where

$$\begin{aligned} A &=_{\text{def}} (\{0\}, \{1\}, \ldots, \{0\}, \{1\}, \{1\}, \{0\}, \ldots, \{0\}, \{0\}, \ldots, \{0\}, \{0\}, \ldots, \{0\}) \\ \mathcal{F} &=_{\text{def}} \langle f_1,\ f_1',\ \ldots,\ f_n,\ f_n',\ e_0,\ e_1,\ \ldots,\ e_n,\ g_0,\ \ldots,\ g_n,\ h_0,\ \ldots,\ h_n \rangle \\ \mathbf{x} &=_{\text{def}} (u_1,\ \overline{u}_1,\ \ldots,\ u_n,\ \overline{u}_n,\ c_0,\ c_1,\ \ldots,\ c_n,\ s_0,\ \ldots,\ s_n,\ z_0,\ \ldots,\ z_n)\ . \end{aligned}$$

The functions $f_1, f_1', \ldots, f_n, f_n'$ are taken from $\mathcal{F}^n_{\text{enum}} = \langle f_1, f_1', \ldots, f_n, f_n' \rangle$, and they are defined on the variables $u_1, \ldots, \overline{u}_n$. The functions $e_i, g_i, h_i$ are defined as follows. For $i \in \{1, \ldots, n\}$, let

$$\begin{aligned} e_0(\mathbf{x}) &=_{\text{def}} c_0 \\ e_i(\mathbf{x}) &=_{\text{def}} e_{i-1}(\mathbf{x}) \cap f_i(\mathbf{x}) \\[1ex] g_0(\mathbf{x}) &=_{\text{def}} s_0 \\ g_i(\mathbf{x}) &=_{\text{def}} (u_i \cap s_i) \cup z_{i-1} \\[1ex] h_0(\mathbf{x}) &=_{\text{def}} \text{val}_{H'}(\mathbf{x}) \\ h_i(\mathbf{x}) &=_{\text{def}} e_i(\mathbf{x}) \cap \begin{cases} (g_i(\mathbf{x}) \cup h_{i-1}(\mathbf{x})) & \text{, if } Q_{n-i+1} = \exists \\ (g_i(\mathbf{x}) \cap h_{i-1}(\mathbf{x})) & \text{, if } Q_{n-i+1} = \forall\,. \end{cases} \end{aligned}$$

Function $\mathrm{val}_{H'}(\mathbf{x})$ will be specified later. To give an idea, this function evaluates $H'$ for the assignment defined by $\mathbf{x}$. More precisely, it shall hold that $1 \in S[h_0](t) \Leftrightarrow H^{(0)}(t) \equiv 1$. We will show that $S[h_n](2^n-1)$ contains 1 if and only if $H \in \mathrm{QBF}$. Observe that $1 \in S[e_i](t)$ for $t \geq 0$ if and only if $\mathrm{bin}_{i-1}(t) = \cdots = \mathrm{bin}_0(t) = 1$ due to Lemma 6.15.

**Claim A** Let $i \in \{1, \ldots, n\}$ and $t_0 \in \mathbb{N}$ such that $1 \in S[e_i](t_0)$. Then, $1 \in S[g_i](t_0)$ if and only if $1 \in S[h_{i-1}](t_0 - 2^{i-1})$.

We show by induction over $t \in \{t_0 - 2^{i-1} + 1, \ldots, t_0\}$ that $1 \in S[g_i](t)$ if and only if $1 \in S[h_{i-1}](t_0 - 2^{i-1})$. Since $1 \notin S[f_i](t_0 - 2^{i-1})$ it holds that $1 \in S[g_i](t_0 - 2^{i-1} + 1) \Leftrightarrow 1 \in S[h_{i-1}](t_0 - 2^{i-1})$. Let $t \in \{t_0 - 2^{i-1} + 1, \ldots, t_0 - 1\}$. $1 \notin S[e_{i-1}](t)$ hence $1 \notin S[h_{i-1}](t)$. We obtain:

$$\begin{aligned} 1 \in S[g_i](t+1) &\Longleftrightarrow 1 \in \Big(S[f_i](t) \cap S[g_i](t)\Big) \cup S[h_{i-1}](t) \\ &\Longleftrightarrow 1 \in S[g_i](t) \\ &\Longleftrightarrow 1 \in S[h_{i-1}](t_0 - 2^{i-1}) \,. \end{aligned}$$

**Claim B** Let $t_0 \in \mathbb{N}$ such that $1 \in S[e_n](t_0)$. Then, $1 \in S[h_n](t_0)$ if and only if $H \equiv 1$.

We show by induction over $i \in \{0, \ldots, n\}$ that for every $t \in \mathbb{N}$ where $1 \in S[e_i](t)$ holds: $1 \in S[h_i](t) \Longleftrightarrow H^{(i)}(t) \equiv 1$. For $i = 0$ the claim holds by definition. Let $i \in \{0, \ldots, n-1\}$. If $Q_{n-i} = \exists$ then

$$\begin{aligned} 1 \in S[h_{i+1}](t) &\Longleftrightarrow 1 \in S[e_{i+1}](t) \cap (S[g_{i+1}](t) \cup S[h_i](t)) \\ &\Longleftrightarrow 1 \in S[g_{i+1}](t) \cup S[h_i](t) \\ &\Longleftrightarrow 1 \in S[h_i](t - 2^i) \cup S[h_i](t) \\ &\Longleftrightarrow H^{(i)}(t - 2^i) \equiv 1 \text{ or } H^{(i)}(t) \equiv 1 \\ &\Longleftrightarrow H^{(i+1)}(t) \equiv 1 \,. \end{aligned}$$

If $Q_{n-i} = \forall$ the proof is similar to the $\exists$-case.

To obtain function $\mathrm{val}_{H'}$ we introduce new variables $\overline{x}_1, \ldots, \overline{x}_n$ to $H'$ and replace every occurrence of $\neg x_i$ by $\overline{x}_i$. (Remember that $\neg$ in quantified Boolean formulas only apply to variables.) Replacing $\vee$ and $\wedge$ by $\cup$ and $\cap$, respectively, and $x_i$ and $\overline{x}_i$ by $u_{n-i+1}$ and $\overline{u}_{n-i+1}$, respectively, we obtain the function $\mathrm{val}_{H'}$ which has the desired property. $S = (\mathcal{F}, A)$ can be computed by an FL-function. Then, it holds that $H \in \mathrm{QBF} \Longleftrightarrow (S, 1) \in \mathrm{M}_{ex}(\cup, \cap)$.

■

We provide a small example that shall illustrate the construction of the proof of Theorem 6.16. We choose the quantified Boolean formula $H$ as

$$H = \exists x_1 \forall x_2 \exists x_3 ((x_1 \wedge \neg x_2) \vee (\neg x_3 \wedge x_1)) \,.$$

This formula obviously evaluates to 1. In Figure 11 the result of the transformation process is presented by means of a circuit representation. The $S^3_{\mathrm{enum}}$ part is not included into the picture to keep the picture readable. Of course, every function must be represented by its own circuit, which can be obtained from the given picture by deleting all irrelevant vertices. The name of a function labels its output vertex. Vertices with the same name are considered identical, which applies to the vertices $u_1, u_2, u_3$. Little squares represent input vertices.

We now turn to recurrent $\{\cup, \oplus, \otimes\}$-systems. We will show that the exact as well as the existential membership problems for recurrent $\{\cup, \oplus, \otimes\}$-systems are in PSPACE. To achieve this we will first show that we can bound by some "slowly growing" function the number of iteration steps that have to be considered for proving that a number can be generated by a given recurrent $\{\cup, \oplus, \otimes\}$-system. As an auxiliary structure we need to find a special *certificate* that proves membership of a given number in the output set of a recurrent $\{\cup, \oplus, \otimes\}$-system in a specified iteration step. Let $S = (\mathcal{F}, A)$ be a recurrent $\{\cup, \oplus, \otimes\}$-system of dimension $n$, $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$, and let $t \in \mathbb{N}$. With each function of $\mathcal{F}$ we associate an arithmetical $\{\cup, \oplus, \otimes\}$-circuit. Consider the following definition:

$$
\begin{aligned}
f_i^{(0)}(\mathbf{x}) &=_{\text{def}} x_i \\
f_i^{(r+1)}(\mathbf{x}) &=_{\text{def}} f_i(f_1^{(r)}(\mathbf{x}), \ldots, f_n^{(r)}(\mathbf{x}))\,, \quad r \geq 0\,.
\end{aligned}
$$

Furthermore, for every $t \geq 0$, let $f^{(t)}(\mathbf{x}) =_{\text{def}} f_n^{(t)}(\mathbf{x})$. We obtain $f^{(t)}$ by $t$-fold and simultaneous superposition. Similarly, we obtain, by glueing together vertices, a circuit representation $C$ of $f^{(t)}$. Labelling the input vertices of $C$ with the corresponding numbers of $A$ results in a $\{\cup, \oplus, \otimes\}$-circuit representation $C_A$ of $S(t) = f^{(t)}(A)$. The *formula representation* $F$ of $S(t)$ is obtained from $C_A$ by *unfolding* circuit $C_A$: Let $g$ denote the output vertex of $C_A$, and let $g'$ and $g''$ be the predecessors of $g$. Let $C' =_{\text{def}} C_A - g$ be a copy of $C_A$ without $g$ where $g'$ is labelled output vertex. Similarly, $C'' =_{\text{def}} C_A - g$ is a copy of $C_A$ without $g$ with output vertex $g''$. Let $F'$ and $F''$ be the results of unfolding $C'$ and $C''$, respectively. Then, $F$ is the disjoint union of $\{g\}$ and $F'$ and $F''$ and arcs from the former predecessors of $g$ go to $g$. An example is shown in Figure 12. The figure shows the result of unfolding the $\{\otimes\}$-circuit of Figure 8. A *computation tree* for $F$ is a maximal subtree of $F$ such that every vertex with label $\cup$ has exactly one predecessor and every other vertex (vertices with labels $\oplus$ or $\otimes$) has two predecessors. If we additionally label vertices of a computation tree with their values, which is for input vertices the label itself, for vertices with only one predecessor the value of the predecessor and for the other vertices the sum or product of the values of the predecessors, then the root vertex has a member of $S(t)$ as value. The root vertex corresponds to the output vertex of $C_A$. Computation trees have been introduced by McKenzie and Wagner to analyse the complexity of MC($\cup, \oplus$). By construction, it is clear that a number is in $S(t)$ if and only if it is the value of the root vertex of a computation tree for $F$. The main observation is that only one number of the set that is attached to a $\cup$-vertex contributes to the result at the root vertex of a computation tree.

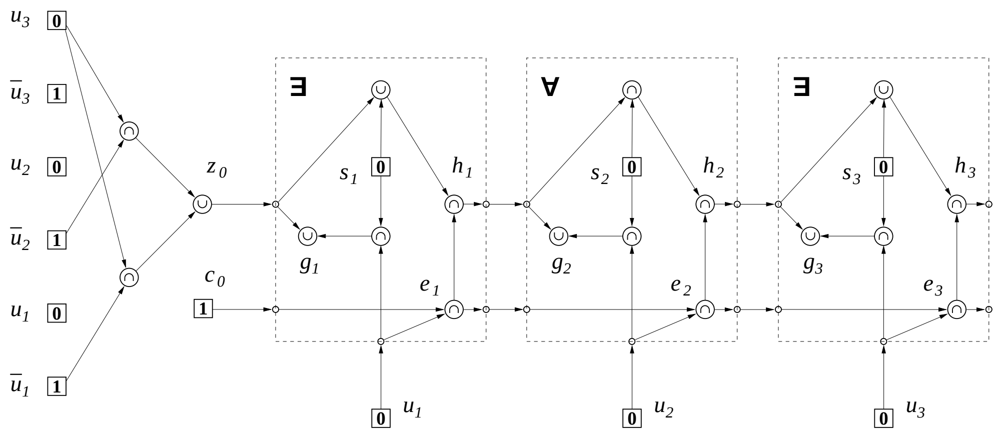


**Figure 11** The result of the reduction of the proof of Theorem 6.16 for formula $H = \exists x_1 \forall x_2 \exists x_3((x_1 \wedge \neg x_2) \vee (\neg x_3 \wedge x_1))$. The quantifiers of $H$ are modelled by the framed parts in reversed order, and the formula part of $H$ can be found on the left side of the picture. The enumeration part $S^3_{\text{enum}}$ is not depicted.

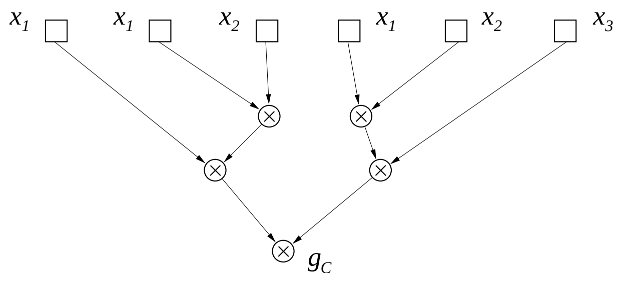


**Figure 12** The result of unfolding a $\{\otimes\}$-circuit representing the function $f(x_1, x_2, x_3) = (x_1)^3(x_2)^2x_3$ .

**Lemma 6.17** *Let $S = (\mathcal{F}, A)$ be a recurrent $\{\cup, \oplus, \otimes\}$-system of dimension $n$. Let $b \in \mathbb{N}$. Then, $b \in [S]$ if and only if there is $t < (b+1) \cdot 2^{n^3}$ such that $b \in S(t)$.*

**Proof:** Let $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$. If $n = 1$ the claim is clear: If $f_1$ contains neither $\oplus$ nor $\otimes$, input equals output. If $f_1$ contains $\oplus$ or $\otimes$, on input greater than 0 it takes at most $b$ iteration steps to know whether $b$ can be generated.

Let $n \geq 2$. If $b \leq 1$ then it suffices to evaluate $S$ in the following way. Input numbers greater than 1 are replaced by 2. In every iteration step $t \geq 1$, instead of using $S_i(t-1)$ as $i$-th input, for every $i \in \{1, \ldots, n\}$, we use $(S_i(t_1 - 1) \cap \{0, 1\}) \cup \{2\}$ as $i$-th input. We observe that, for $b \leq 1$, $b \in S(t)$ if and only if $b$ is contained in the output set of the $t$-th described iteration step. Since there are only eight possible sets for each input, the number of iteration steps to decide $(S, b) \in \mathrm{M}_{ex}(\cup, \oplus, \otimes)$ can be bounded above by $8^n = 2^{3n} < 2^{n^3}$. Remember the argumentation of the proof of Theorem 6.13 and the definition of similar tuples.

Let $b \geq 2$. We assume that no function of $\mathcal{F}$ is of the form $\varphi(\mathbf{x}) = x_i$. Otherwise, we add an $(n{+}1)$-th component $(f_{n+1}(\mathbf{x}') =_{\mathrm{def}} x_1 \oplus x_{n+1}, b{+}1)$ to $S$, $\mathbf{x}' = (x_1, \ldots, x_{n+1})$, and replace every function $f_i(\mathbf{x}) = x_j$ by $f'_i(\mathbf{x}') =_{\mathrm{def}} x_j \cup x_{n+1}$. Obviously, this does not effect the question $b \in [S]$ and is only of technical advantage. Let $t \geq 0$ be smallest possible such that $b \in S(t)$. Let $F$ be the formula representation of $S(t)$, and let $T$ be a computation tree for $F$ with root value $b$. The *reduced* computation tree $B$ of $T$ is the maximal subtree of $T$ containing the root vertex of $T$ and that does not contain vertices with value greater than 0 in a subtree with root vertex labelled with $\otimes$ and with value 0. In other words, if $u$ is a vertex in $B$ that is labelled with $\otimes$ and has value 0, then all paths in $B$ from input vertices to $u$ contain only vertices with value 0. We define *levels* of $B$ as follows. Each vertex of $B$ that corresponds to the output vertex of the circuit representation of some function of $\mathcal{F}$ is labelled with the index of the corresponding function; the input vertices of $B$ are labelled with the index of the corresponding variable. Note that some vertices now have two kinds of labels: numbers from $A$ or $\cup, \oplus, \otimes$ and indices $1, \ldots, n$. Then, $L_r$ contains exactly those vertices that are labelled with an index and that lie each on a leaf-root path in $B$ with exactly $r$ index-labelled vertices preceding it. Leaves or input vertices of $B$ are exactly the vertices in $L_0$ whereas the root vertex of $B$ is the only member of $L_t$. Observe that every leaf-root path in $B$ passes the same number of index-labelled vertices. Note the following correspondence: if some vertex from $L_r$ is labelled with index $i$ and has value $a$, then $a \in S_i(r)$.

By our definition of a reduced computation tree the value of a vertex in $B$ is not smaller than the value of any of its predecessors. Let $u$ and $v$ be vertices in $B$ with the same value such that there is a path from $v$ to $u$. Then, every vertex on the $v,u$-path has the same value as $u$ and $v$. Suppose there are $r_1, r_2 \in \mathbb{N}$ such that $r_1 \geq 2^{n^3} + r_2$ and for each vertex $u$ in $L_{r_1}$ there is a vertex $v \in L_{r_2}$ in the subtree of $B$ rooted by $u$ such that $u$ and $v$ have the same value. Such vertices constitute pairs. If $u$ has value greater than 1, there is only one such pair for $u$. Let $(u, v)$ be a pair with value of $u$ greater than 1. Consider the $v,u$-path $P$. Let $w$ be a vertex on $P$ different from $v$. If $w$ has label $\oplus$ then its predecessor that does not lie on $P$ has value 0, if $w$ has label $\otimes$ the respective predecessor has value 1. Let $(u', v')$ be another pair such that $u$ and $u'$ as well as $v$ and $v'$ have the same index-labels, respectively. Then, we can replace the subtree rooted by $u'$ by a copy of the subtree rooted by $u$, replace the subtree rooted by $v$ in the copy by the subtree rooted by $v'$ (see Figure 13) and obtain a reduced computation tree for $F$ with root value $b$. We repeat this procedure for other pairs as long as possible and obtain a reduced computation tree $B^*$ for $F$ with root value $b$ that contains at most $n^2$ different subtrees between $L_{r_1}$ and $L_{r_2}$ rooted by vertices with values greater than 1. To each level $L_r$ we assign a tuple indicating for every $i \in \{1, \dots, n\}$ whether a vertex in $L_r$ has label $i$ and value 0 or value 1 and for every pair $(i, j) \in \{1, \dots, n\}^2$ the index of the vertex in $L_r$ of $B^*$ that is passed by the path between a vertex in $L_{r_2}$ with label $j$ and value greater than 1 and a vertex in $L_{r_1}$ with label $i$. There are at most $2^{2n} \cdot n^{n^2} \leq 2^{n^3}$ different tuples possible. It follows that there must be two levels $L_{s_1}$ and $L_{s_2}$, $r_1 \leq s_1 < s_2 \leq r_2$, with the same assigned tuple. Then, we can delete levels $L_{s_1}, \dots, L_{s_2-1}$ and build a reduced computation tree with root value b that proves $b \in S(t - (s_2 - s_1))$, which is a contradiction to the assumption that $t$ is smallest possible.

Now, we can conclude the proof by first observing the following important properties:

(1) $B$ contains a leaf with value greater than 0

(2) for every $r \geq 2^{n^3}$ there is a vertex $u$ in $L_r$ such that every vertex in $L_{r-2^{n^3}}$ in the subtree rooted by $u$ has a smaller value than the value of $u$, and $u$ has value greater than 1. Note that, if $u$ has value 1, there is a path from an input vertex to $u$ that contains only vertices with value 1.

If $\sum^{\geq 2} L_r$ denotes the sum of the values of the vertices in level $L_r$ with value greater than 1, it holds for every $r < t$ that $\sum^{\geq 2} L_r \leq \sum^{\geq 2} L_{r+1}$. It follows that $\sum^{\geq 2} L_{2^{n^3}} \geq 2$, $\sum^{\geq 2} L_{2 \cdot 2^{n^3}} \geq 3$, and so on, so that $t \leq b \cdot 2^{n^3}$.

■

From the just proved lemma, it follows that $\mathcal{M} = \{\cup, \oplus, \otimes\}$ is a maximal set of operators for which the number of iteration steps for recurrent $\mathcal{M}$-systems to generate numbers can be bounded. $\mathrm{M}_{ex}(\cup, \cap, \oplus, \otimes)$ and $\mathrm{M}_{ex}(\cup, \cap, ^-, \oplus, \otimes)$ are undecidable, as we will see in the next chapter. At first glance, one might argue that undecidability of the latter problem is due to (the expected) undecidability of the non-iterated variant $\mathrm{MC}(\cup, \cap, ^-, \oplus, \otimes)$. However, every bound for the general problem serves as bound for a restricted variant, hence $\mathrm{M}_{ex}(\cup, \cap, \oplus, \otimes)$ would be decidable due to decidability of $\mathrm{M}_{tm}(\cup, \cap, \oplus, \otimes)$.

The problems $\mathrm{M}_{tm}(\mathcal{O})$ for $\mathcal{O} \subseteq \{\cup, \cap, ^-, \oplus, \otimes\}$ can be considered similar to the problems $\mathrm{MC}(\mathcal{O})$ with succinct input representation. For most of the previously studied problems succinctness led to an increasement of complexity. With this phenomenon in mind it is surprising that we can show that $\mathrm{M}_{tm}(\cup, \oplus, \otimes)$ is solvable in polynomial space. It is known that $\mathrm{MC}(\cup, \oplus, \otimes)$ is complete for polynomial space

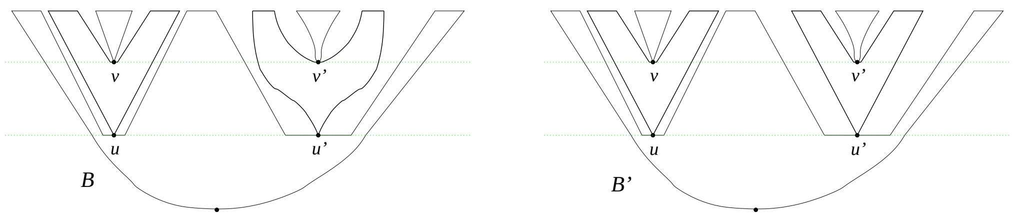


**Figure 13** The construction of Lemma 6.17. $B'$ is the reduced computation tree that emerged from $B$ by the replacement operation.

(Theorem 5.7 and [87]). To give a short introduction to the main idea of the following proof, consider some recurrent $\{\oplus\}$-system $S$, and let $F$ be the formula representation of $S(t)$ for some $t \geq 0$. Then, $F$ is also a computation tree, and the value of the root vertex is determined by the number of paths from leaves to the root vertex (see also the discussion in connection with Lemma 6.3). If we additionally have $\cup$- and $\otimes$-vertices, not all paths in the formula representation contribute to the result.

**Theorem 6.18** $\mathrm{M}_{tm}(\cup, \oplus, \otimes)$ *is in* PSPACE.

**Proof:** Let $(S, t, b)$ be an instance of $\mathrm{M}_{tm}(\cup, \oplus, \otimes)$ where $S = (\mathcal{F}, A)$ is a recurrent $\{\cup, \oplus, \otimes\}$-system of dimension $n$, $\mathcal{F} = \langle f_1, \ldots, f_n \rangle$ and $A = (a_1, \ldots, a_n)$. Let $C_1, \ldots, C_n$ be the circuit representations of $f_1, \ldots, f_n$, respectively. Similar to the construction in the proof of Lemma 6.17, we assume that no circuit has output vertex that is an input vertex. In that proof, we have seen that, if $b \leq 1$, it suffices to consider only subsets of $\{0, 1, 2\}$ as input and output sets of the functions (also remember the definition of tuples similar with respect to some given number in the proof of Theorem 6.13). Hence, in polynomial space, $b \in S(t)$ can be decided by a straightforward evaluation.

Let $b \geq 2$. Let $b \in S(t)$. Let $F$ be the formula representation of $S(t)$ obtained by unfolding the circuit representation of $S(t)$, and let $T$ be a computation tree for $F$ with root value $b$. Observe the following: any leaf-root path that does not contain a vertex with value 0 contains at most $\lfloor \log b \rfloor$ $\otimes$-vertices without predecessors with value 1. We will give an algorithm that verifies the existence of a computation tree for $F$ with root value $b$. The main idea is as follows. Let $B'$ emerge from $T$ by replacing every $\otimes$-vertex of $T$ by an input vertex of the same value. (Delete all vertices that are not accessible from the root.) Obviously, the value of every vertex $u$ in $B'$ is the sum of the input vertices in the subtree rooted at $u$ and is equal to the value of the corresponding vertex of $T$. Now, obtain $B$ from $B'$ by removing all vertices with value 0. Then, there are at most $b$ leaves in $B$, and $\oplus$-vertices may have only one predecessor.

A vertex of $B$ is *marked* if it corresponds to a vertex that emerged in the circuit representation of $S(t)$ from glueing together an input and an output vertex. These marked vertices are additionally labelled with the number of the original input vertex. (Recall Figure 4: the right circuit contains two such vertices that emerged from a glue operation, and they would be labelled with 1 and 2.) We consider the levels of $B$ defined in the same way as in the proof of Lemma 6.17, i.e., marked vertex $u$ belongs to level $r$, denoted as $L_r$, $r \in \{0, \ldots, t\}$, if and only if $u$ is contained in the subtree of $B$ of exactly $t - r + 1$ marked vertices. The root vertex of $B$ solely forms $L_t$. With $L_r$ we associate an $(n+1)$-tuple $(\nu_1^r, \ldots, \nu_n^r, \sigma^r)$ where $\nu_i^r$, $i \in \{1, \ldots, n\}$, denotes the number of marked vertices in $L_r$ with label $i$ and $\sigma^r$ is the sum of the values of all marked vertices in $L_r$. Note that this sum may be smaller than $b$ since subtrees with root vertex a $\otimes$-vertex were cut. Our algorithm will iteratively generate such tuples associated to levels starting from level $t$ and verify that the difference between the values of the last components (the *$\sigma$-components*) of the tuples associated to consecutive levels is the sum of the values of appropriate computation trees (those that were cut). The tuple $(0,\ldots,0,1,b)$ is associated to level $t$. Let tuple $(\nu_1^r,\ldots,\nu_n^r,\sigma^r)$, associated to level $r>0$, be known. We describe how to find a tuple for level $r-1$. Remember that no vertex in $B$ has value 0. Let $i\in\{1,\ldots,n\}$ be such that $\nu_i^r>0$. Decrease $\nu_i^r$ by 1. The algorithm guesses how much a copy of a computation tree of the unfolded circuit representation $C_i$ of $f_i$ contributes to $\tau^{r-1} =_{\text{def}} (\nu_1^{r-1},\ldots,\nu_n^{r-1},\sigma^{r-1})$, i.e., which vertices from $L_{r-1}$ are contained in a subtree rooted by some vertex from $L_r$ marked with label $i$. Let the vertices of $C_i$ be ordered arbitrarily. In polynomial space we can examine the paths of $C_i$ one after the other. If a path contains a $\cup$-vertex, the subtree of exactly one predecessor must be considered. If a path contains a $\otimes$-vertex the value of the $\otimes$-vertex at shortest distance to the output vertex must be determined. This vertex then corresponds to a leaf of $B'$ that replaced a $\otimes$-vertex in $T$. This routine is described in the next paragraph. For each $\oplus$-vertex $u$ it must be guessed whether it has value 0 or greater than 0. The former case can be verified in polynomial space as follows. Let $f'$ be a sub-function of $f_i$ that computes the value of $u$. (This function is represented by a sub-circuit of $C_i$.) The algorithm simply verifies $(S',r,0)\in \mathrm{M}_{tm}(\cup,\oplus,\otimes)$ where $S'$ is obtained from $S$ by adding function $f'$ as $(n+1)$-th function. If $u$ is decided to have value 0 then the subtree rooted at $u$ does not have to be considered and does not contribute to $\tau^{r-1}$. If $u$ has value greater than 0 all leaves with value greater than 0 in its subtree contribute to $\tau^{r-1}$. The algorithm repeats this procedure until there is no $i\in\{1,\ldots,n\}$ such that $\nu_i^r>0$. Set $\sigma^{r-1} =_{\text{def}} \sigma^r$. Tuple $\tau^{r-1}$ is generated and the algorithm proceeds with this tuple until the tuple associated to level 0 is obtained. If $\Sigma_{i=1}^n \nu_i^0\cdot a_i=\sigma^0$ then accept. If $b\notin S(t)$, the existence of no reduced computation tree with root value $b$ can be proved. The above described part of our algorithm needs only (nondeterministic) polynomial space.

It remains to show how to verify the value of a $\otimes$-vertex $u$. Similar to the case of a $\oplus$-vertex with value 0, we can verify in polynomial space whether $u$ has value 0 or 1. Let $u_1$ and $u_2$ be the predecessors of $u$. If $u$ has value $p>1$ then $u_1$ must have value $p_1>0$ and $u_2$ must have value $p_2>0$ such that $p=p_1\cdot p_2$. It can be verified in polynomial space as described above whether $p_1=1$ or $p_2=1$ can be chosen. If one is true, $u$ can be treated as $\oplus$-vertex with one predecessor having value 0. Otherwise reduce $\sigma^r$ by $p$ and the algorithm verifies $(S_1',r,p_1)\in \mathrm{M}_{ex}(\cup,\oplus,\otimes)$ and then $(S_2',r,p_2)\in \mathrm{M}_{ex}(\cup,\oplus,\otimes)$, where $S_1'$ and $S_2'$ emerge from $S$ by adding as $(n+1)$-th component the subfunctions of $f_i$ computing the values of $u_1$ and $u_2$, respectively. Hence, the space needed by the algorithm depends on the number of such verifications that are carried out at the same time. It holds that every additional verification process corresponds to some $\otimes$-vertex and these vertices are contained in one leaf-root path of $F$. Then, there can be at most $\lfloor \log b\rfloor$ verification processes running at the same time, since $2\le p_1,p_2\le \frac{p}{2}$. This gives a polynomial space upper complexity bound for the total algorithm.

■

Lemma 6.17 and Theorem 6.18 are highly involved and complex results. But now, it is easy to prove the final results of this section, using the following well-known theorem about the complexity of the quantified Boolean formula problem.

**Theorem 6.19 (Stockmeyer and Meyer, [79])** *QBF is* PSPACE*-complete.*

**Theorem 6.20** $\mathrm{M}_{ex}(\cup,\cap)$ *and* $\mathrm{M}_{ex}(\cup,\cap,^-)$ *as well as* $\mathrm{M}_{ex}(\oplus,\otimes)$*,* $\mathrm{M}_{ex}(\cup,\oplus)$*,* $\mathrm{M}_{ex}(\cup,\otimes)$ *and* $\mathrm{M}_{ex}(\cup,\oplus,\otimes)$ *are* PSPACE*-complete.*

**Proof:** We first show that $\mathrm{M}_{ex}(\cup,\cap,^-)$ is contained in PSPACE. Let $(S,b)$ be an instance of $\mathrm{M}_{ex}(\cup,\cap,^-)$ where $S=(\mathcal{F},A)$ is a recurrent $\{\cup,\cap,^-\}$-system of dimension $n$. There are exactly $2^n$ possible configurations of $\mathcal{F}(t)$ with respect to $b$, since for every input set $B$, it is only important to know whether $b\in B$ or $b\notin B$. So, we can consider input sets only as subsets of $\{b\}$, and $\mathcal{F}(t)$ for every $t\geq 0$ contains only $\emptyset$ and $\{b\}$ as entries. Start the evaluation process of $S$. If a configuration appears that has already been encountered there will be no new configuration of $S$ with respect to $b$. After $2^n$ evaluation steps all possible configurations that can be generated by $S$ have been encountered. Furthermore, the results of every iteration step can be written down using only $n$ bits, so that $\mathrm{M}_{ex}(\cup,\cap,^-)$ can be computed using only polynomial space, i.e., $\mathrm{M}_{ex}(\cup,\cap,^-)$ is in PSPACE. Containment in PSPACE of all other problems is due to Lemmata 5.2 and 6.17 and Theorem 6.18.

For hardness, we show that $\mathrm{M}_{ex}(\cup,\cap)$ reduces to $\mathrm{M}_{ex}(\oplus,\otimes)$, $\mathrm{M}_{ex}(\cup,\oplus)$ and $\mathrm{M}_{ex}(\cup,\otimes)$. Let $(S,b)$ be an instance of $\mathrm{M}_{ex}(\cup,\cap)$ where $S=(\mathcal{F},A)$ is a recurrent $\{\cup,\cap\}$-system of dimension $n$ and $\mathcal{F}=\langle f_1,\ldots,f_n\rangle$. Let $f'_i$, $i\in\{1,\ldots,n\}$, emerge from $f_i$ by replacing every $\cap$ by $\oplus$ and every $\cup$ by $\otimes$. Let $A'$ emerge from $A$ by replacing $b$ by 0 and every other number by 1. Let $S'=_{\text{def}}(\langle f'_1,\cdots,f'_n\rangle,A')$. It holds that $x+y=0\Leftrightarrow x=y=0$ and $x\cdot y=0\Leftrightarrow x=0$ or $y=0$. By induction, it follows that $b\in S(t)$ if and only if $0\in S'(t)$, $t\in\mathbb{N}$. Hence, $(S,b)\in\mathrm{M}_{ex}(\cup,\cap)$ if and only if $(S',0)\in\mathrm{M}_{ex}(\oplus,\otimes)$. Similarly, $\mathrm{M}_{ex}(\cup,\cap)$ reduces to $\mathrm{M}_{ex}(\cup,\oplus)$ and $\mathrm{M}_{ex}(\cup,\otimes)$. Hardness of all problems follows by Lemma 5.2 and Theorems 6.19 and 6.16.

■

**Corollary 6.21** $\mathrm{M}_{tm}(\cup,\cap)$ *and* $\mathrm{M}_{tm}(\cup,\cap,^-)$ *as well as* $\mathrm{M}_{tm}(\oplus,\otimes)$*,* $\mathrm{M}_{tm}(\cup,\oplus)$*,* $\mathrm{M}_{tm}(\cup,\otimes)$ *and* $\mathrm{M}_{tm}(\cup,\oplus,\otimes)$ *are* PSPACE*-complete.*

**Proof:** Let $(S,b)$ be an instance of $\mathrm{M}_{ex}(\cup,\cap)$ where $S$ is a recurrent $\{\cup,\cap\}$-system of dimension $n$. Remember that $(S,b)\in\mathrm{M}_{ex}(\cup,\cap)$ if and only if $(S,t,b)\in\mathrm{M}_{tm}(\cup,\cap)$ for some $t\leq 2^n$ (proof of Theorem 6.20). Let $\mathbf{x}=_{\text{def}}(x_1,\ldots,x_{n+1})$, and let $b'$ be the smallest number such that $b'\neq b$. Add component $(f_{n+1}(\mathbf{x})=_{\text{def}}x_n\cup x_{n+1},b')$ to $S$ and obtain $S'$. Then, $(S,b)\in\mathrm{M}_{ex}(\cup,\cap)$ if and only if $(S',2^n,b)\in\mathrm{M}_{tm}(\cup,\cap)$. Since $\mathrm{M}_{tm}(\cup,\cap)$ reduces to all other problems, the statement holds. Note that containment in PSPACE of all problems is due to the proof of Theorem 6.20 and due to Lemma 5.2 and Theorem 6.18.

■

## 6.4 Further polynomial space hard membership problems

About half the number of exact and existential membership problems for recurrent systems has not yet been treated. Three of these problems are considered in the next chapter, that discusses undecidability results. Two further problems are considered in the next section. For the remaining problems, that we treat in this section, we cannot give full characterisations of their complexities. In other words, upper or lower complexity bounds that will be given here are not tight. At least, all problems of this section share the loose property of being PSPACE-hard. We begin the discussion by proving this property.

**Lemma 6.22** *(1)* $\mathrm{M}_{tm}(\cup,\cap) \leq_m^{\mathrm{L}} \mathrm{M}_{tm}(^-,\oplus)$ *and* $\mathrm{M}_{ex}(\cup,\cap) \leq_m^{\mathrm{L}} \mathrm{M}_{ex}(^-,\oplus)$.
*(2)* $\mathrm{M}_{tm}(\cup,\cap) \leq_m^{\mathrm{L}} \mathrm{M}_{tm}(^-,\otimes)$ *and* $\mathrm{M}_{ex}(\cup,\cap) \leq_m^{\mathrm{L}} \mathrm{M}_{ex}(^-,\otimes)$.

**Proof:** For proving statement (1), let $S = (\mathcal{F}, A)$ be a recurrent $\{\cup,\cap\}$-system of dimension $n$ and $\mathcal{F} = \langle f_1,\ldots,f_n\rangle$. Let $f_1',\ldots,f_n'$ be the functions that emerge from $f_1,\ldots,f_n$ by replacing every $\cap$ by $\oplus$ and every $B\cup C$ by $\overline{\overline{B}\oplus\overline{C}}$. It is clear that the replacement for $\cup$-vertices describes a 4-vertex circuit. The following two equivalences hold. Let $B, C \subseteq \mathbb{N}$.

$$0 \in B\cap C \iff 0 \in B\oplus C \qquad \text{and}$$

$$\begin{aligned} 0 \in B\cup C &\iff 0 \notin \overline{B} \text{ or } 0 \notin \overline{C} \\ &\iff 0 \notin (\overline{B}\oplus\overline{C}) \\ &\iff 0 \in \overline{\overline{B}\oplus\overline{C}}\,. \end{aligned}$$

Let $b \in \mathbb{N}$. Let $A'$ emerge from $A$ by replacing $b$ by 0 and every other number by 1. Let $S' =_{\mathrm{def}} (\langle f_1',\ldots,f_n'\rangle, A')$. It then holds for every $t' \geq 0$: $b \in S(t') \Leftrightarrow 0 \in S'(t')$, i.e., $(S,t,b) \in \mathrm{M}_{tm}(\cup,\cap)$ if and only if $(S',t,0) \in \mathrm{M}_{tm}(^-,\oplus)$. Equally, $(S,b) \in \mathrm{M}_{ex}(\cup,\cap)$ if and only if $(S',0) \in \mathrm{M}_{ex}(^-,\oplus)$.

For statement (2), we use a similar reduction. Let $S$ be a recurrent $\{\cup,\cap\}$-system. Obtain $S''$ from $S$ by replacing every $\cup$ by $\otimes$ and every $B\cap C$ by $\overline{\overline{B}\otimes\overline{C}}$. Let $B, C \subseteq \mathbb{N}$ be non-empty. Then,

$$\begin{aligned} 0 \in B\cup C &\iff 0 \in B\otimes C \\ 0 \in B\cap C &\iff 0 \in \overline{\overline{B}\otimes\overline{C}}\,. \end{aligned}$$

Let $b \in \mathbb{N}$. When we replace every $b$ in $A$ by 0 and every other number by 1, this yields $A''$, the initial number set of $S''$, and it holds, for every $t' \geq 0$, $b \in S(t') \Leftrightarrow 0 \in S''(t')$. One special case in this construction has to be verified, in fact, that $\emptyset$ cannot be computed, to fulfill the prerequisites of the equivalences above. Remember that, during the first iteration step, the functions of $S''$ are applied to singleton sets only $\{0\}$ and $\{1\}$. By induction, it remains to prove that every possible computed set does not contain 0 and 1. This is clear for the complementation operation. But since

$1 \in B \otimes C$ for every $B, C \subseteq \mathbb{N}$ if and only if $1 \in B \cap C$, the required property also holds for $\otimes$. Hence, $(S, t, b) \in \mathrm{M}_{tm}(\cup, \cap)$ if and only if $(S'', t, 0) \in \mathrm{M}_{tm}(^-, \otimes)$, and $(S, b) \in \mathrm{M}_{ex}(\cup, \cap)$ if and only if $(S'', 0) \in \mathrm{M}_{ex}(^-, \otimes)$.

■

**Theorem 6.23** *$\mathrm{M}_{tm}(^-, \oplus)$, $\mathrm{M}_{tm}(\cup, \cap, \oplus)$ and $\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus)$ as well as $\mathrm{M}_{tm}(^-, \otimes)$, $\mathrm{M}_{tm}(\cup, \cap, \otimes)$ and $\mathrm{M}_{tm}(\cup, \cap, ^-, \otimes)$ as well as $\mathrm{M}_{tm}(\cap, \oplus, \otimes)$ and $\mathrm{M}_{tm}(^-, \oplus, \otimes)$ are* PSPACE*-hard.*

**Proof:** $\mathrm{M}_{tm}(\cup, \cap)$ is PSPACE-hard due to Corollary 6.21. Due to Lemma 6.22, it follows that all stated problems except for $\mathrm{M}_{tm}(\cap, \oplus, \otimes)$ are PSPACE-hard. Hardness of $\mathrm{M}_{tm}(\cap, \oplus, \otimes)$ directly follows from PSPACE-hardness of $\mathrm{M}_{tm}(\oplus, \otimes)$ (equally Corollary 6.21).

■

**Theorem 6.24** *$\mathrm{M}_{ex}(^-, \oplus)$, $\mathrm{M}_{ex}(\cup, \cap, \oplus)$ and $\mathrm{M}_{ex}(\cup, \cap, ^-, \oplus)$ as well as $\mathrm{M}_{ex}(^-, \otimes)$, $\mathrm{M}_{ex}(\cup, \cap, \otimes)$ and $\mathrm{M}_{ex}(\cup, \cap, ^-, \otimes)$ as well as $\mathrm{M}_{ex}(\cap, \oplus, \otimes)$ are* PSPACE*-hard.*

**Proof:** Using Lemma 6.22 and Theorem 6.20, the proof is similar to the proof of Theorem 6.23.

■

We have proved lower bounds. Now, we show some complexity upper bounds results. At first, we consider two results that do not trivially follow from previous results (but the techniques are easy).

**Lemma 6.25** *$\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus)$ is in* EXP.

**Proof:** Let $(S, t, b)$ be an instance of $\mathrm{M}_{tm}(\cup, \cap, ^-, \oplus)$. The main observation that has to be made here is that $(S, t, b) \in \mathrm{M}_{tm}(\cup, \cap, ^-, \oplus)$ if and only if $(S, b, t, b) \in \mathrm{aM}_{tm}(\cup, \cap, ^-, \oplus)$. This is the case since numbers larger than $b$ do not contribute to the results with respect to the question $b \in S(t)$. Since $\mathrm{aM}_{tm}(\cup, \cap, ^-, \oplus)$ is in EXP due to Lemma 5.3, we conclude the proof.

■

**Lemma 6.26** *$\mathrm{M}_{ex}(\cup, \cap, ^-, \oplus)$ is in* EXPSPACE.

**Proof:** Let $(S, b)$ be an instance of $\mathrm{M}_{ex}(\cup, \cap, ^-, \oplus)$ where $S$ is of dimension $n$. The proof of Lemma 6.25 shows that only subsets of $\{0, \ldots, b\}$ need to be considered. Since there are only $(2^{b+1})^n = 2^{n \cdot (b+1)}$ $n$-tuples over $\mathbb{N}$ that are different with respect to the question $b \in [S]$, it follows that $b \in [S]$ if and only if there is $t < 2^{n \cdot (b+1)}$ such that $b \in S(t)$. In exponential space, $(S, t, b) \in \mathrm{M}_{tm}(\cup, \cap, ^-, \oplus)$ can be verified for every $t < 2^{n \cdot (b+1)}$. Since input length is determined by $(S, b)$, it takes exponential space to write down these numbers, and together with the proof of Lemma 6.25, we conclude the proof.

■

It would be nice to have a similar result for the existential membership problem for recurrent $\{\cup, \cap, ^-, \otimes\}$-systems. However, the equivalence

$$b \in B \otimes C \iff b \in (B \cap \{0, \dots, b\}) \otimes (C \cap \{0, \dots, b\})$$

does not hold for $b = 0$, and this non-monotone behaviour of multiplication causes most problems for analysing corresponding membership problems. In case of exact membership problems, we obtain complexity upper bounds by transfering results for the non-iterated membership problems MC to our problems.

**Lemma 6.27** *(1)* $\mathrm{M}_{tm}(\cap, \otimes)$ *is in* EXP.

*(2)* $\mathrm{M}_{tm}(\cap, \oplus, \otimes)$ *is in* coNEXP.

*(3)* $\mathrm{M}_{tm}(\cup, \cap, ^-, \otimes)$ *is in* EXPSPACE.

*(4)* $\mathrm{M}_{tm}(\cup, \cap, \oplus, \otimes)$ *is in* 2-NEXP.

**Proof:** All these problems are obtained using the following construction. For $S$ a recurrent system and $b$ and $t$ numbers, the question $b \in S(t)$ is equal to the question $b \in S'(1)$ where $S'$ emerges from $S$ by $t$-fold superposition of the involved circuits, i.e., a single circuit is constructed. If $\nu$ denotes the sum of the numbers of vertices of the circuits of $S$, $S'(1)$ is represented by a circuit of $t \cdot \nu$ vertices, i.e., by a circuit of size exponential in the size of instances of exact membership problems. Applying Theorem 5.7, we obtain the claimed complexity bounds.

■

**Corollary 6.28** $\mathrm{M}_{ex}(\cup, \cap, ^-, \otimes)$ *and* $\mathrm{M}_{ex}(\cup, \cap, \oplus, \otimes)$ *are computable.*

**Proof:** For every recurrent system $S$ and every number $b$, $b \in [S]$ if and only if there is $t \geq 0$ such that $b \in S(t)$. Since $\mathrm{M}_{tm}(\cup, \cap, ^-, \otimes)$ and $\mathrm{M}_{tm}(\cup, \cap, \oplus, \otimes)$ are decidable due to Lemma 6.27, the mentioned existential membership problems are computable.

■

## 6.5 Two unclassified membership problems

Two membership problems have not yet been considered. Actually, this is not completely true, since for one of these problems, we have already proved an upper bound. The problems, that we speak of here, are the exact and existential membership problem for recurrent $\{\cap, \otimes\}$-systems. We recall what we know so far about the complexities of both problems:

(1) $\mathrm{M}_{tm}(\cap, \otimes)$ is NL-hard by Theorem 6.4 and contained in EXP by Lemma 6.27

(2) $\mathrm{M}_{ex}(\cap, \otimes)$ is NP-hard by Theorem 6.13 and computable by Corollary 6.28.

We observe huge gaps between upper and lower bounds. However, the author strongly believes that, similar to recurrent $\{\cap, \oplus\}$-systems, the complexity upper bounds can be reduced close to the cited lower bounds. In case of $\mathrm{M}_{ex}(\cap, \otimes)$, this mainly depends on small values for the number of iteration steps that have to be performed for deciding whether a queried number can be generated. In case of $\mathrm{M}_{tm}(\cap, \otimes)$, Corollary 6.7 describes a possible way. In fact, it seems that it suffices to solve the emptiness problem for recurrent $\{\cap, \otimes\}$-problems. However, in contrast to recurrent $\{\cap, \oplus\}$-systems, a partial solution of $\textsc{Empty}(\cap, \otimes)$ would not immediately imply a solution for $\mathrm{M}_{tm}(\cap, \otimes)$.

# Chapter 7
# Undecidable membership problems

A strong motivation for the study of (existential) membership problems for recurrent $\{\cup, \cap, ^-, \oplus, \otimes\}$-systems was the major open question for membership problems for arithmetical $\{\cup, \cap, ^-, \oplus, \otimes\}$-circuits: Is $\mathrm{MC}(\cup, \cap, ^-, \oplus, \otimes)$ decidable? As we have seen decidability of exact membership problems for restricted recurrent systems translates directly into decidability for the corresponding circuit problem, and vice versa. So far, there has not yet been a proof of undecidability of any of these problems. Hence, a first step to roughly partition problems into classes of decidable and undecidable problems is the consideration of this question for the existential membership problems.

Interestingly, we can show undecidability for existential membership problems. Precisely, undecidability can and will be shown for $\mathrm{M}_{ex}(\cup, \cap, \oplus, \otimes)$ and $\mathrm{M}_{ex}(^-, \oplus, \otimes)$. The exact membership problem corresponding to the former problem is trivially decidable, since only finite sets are involved. (It is even in the "low" complexity class 2-NEXP.) Decidability of the exact membership problem for recurrent $\{^-, \oplus, \otimes\}$-systems, however, has not yet been proved or disproved.

Our undecidability proofs are based on a reduction from the DIOPHANTINE problem. This problem is also known as *Hilbert's 10th problem*, which asks for an algorithm deciding whether a given Diophantine equation has a natural (alternatively: integer) solution [43]. About seventy years after Hilbert's famous speech, Matiyasevich proved undecidability of DIOPHANTINE. Our reduction will be done in several steps. The main problem is to generate all $k$-tuples over the natural numbers. We will first give an enumeration algorithm, and starting from this, we will define recurrent systems that are assembled to realise the enumeration of all $k$-tuples. The major problem that we will have to solve is to realise subtraction. When proving PSPACE-hardness of $\mathrm{M}_{ex}(\cup, \cap)$ in the previous chapter, we realised a related enumeration algorithm: generating all binary $k$-tuples.

## 7.1 Enumerating $k$-tuples of natural numbers

We want to decide whether a given Diophantine equation has a natural solution by using recurrent systems. The idea is to try every possible input for the equation and to check whether it is a solution. So, it is necessary to generate all inputs, i.e., all $k$-tuples over the set of natural numbers, where $k$ denotes the number of variables in the equation. A simple algorithm is realised by $k$ nested `for`-loops generating, in round $r$, all $k$-tuples between $(0, \ldots, 0)$ and $(r, \ldots, r)$. It is clear that every single tuple is generated infinitely often by this algorithm. We choose a more sophisticated approach, since we wish to apply an algorithm that permits generation of every $k$-tuple exactly once. We define an enumeration algorithm in two steps: first, we define an auxiliary algorithm, that also generates tuples but of only a restricted form, and second, we derive from the auxiliary algorithm the desired enumeration algorithm.

The auxiliary generation algorithm shall generate only monotone tuples. This means: a tuple $(b_1, \ldots, b_k)$ is *monotone* if $b_1 \leq \cdots \leq b_k$. We can order monotone $k$-tuples in the following way: $(a_1, \ldots, a_k) < (b_1, \ldots, b_k)$ if and only if there is $i \in \{1, \ldots, k\}$ such that $a_i < b_i$ and $a_j = b_j$ for all $j \in \{i{+}1, \ldots, k\}$. So, the order relation of two tuples is determined by the rightmost component in which both tuples differ. Certainly, we have defined a lexicographic order on the set of monotone $k$-tuples over $\mathbb{N}$. Our algorithm will generate all monotone $k$-tuples over $\mathbb{N}$ in the order just defined. The algorithm is called $\texttt{Generate}_k$, and it is given in Figure 14. In every iteration step, a $k$-tuple is output. We consider tuple $(0, \ldots, 0)$ to be output in iteration step 0 (algorithm line 4). Before we prove the result for $\texttt{Generate}_k$, we give a table containing the outputs of the first iteration steps of $\texttt{Generate}_3$.

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | – | 3 | 2 | 1 | 3 | 2 | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 2 | 1 | 1 | 2 | 1 | 1 | 1 | 3 | 2 | 1 | 2 | 1 |
| $b_3$ | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| $b_2$ | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 0 | 1 | 1 | 2 | 2 |
| $b_1$ | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 2 | 0 | 0 | 1 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 0 | 1 | 0 | 1 |

**Lemma 7.1** *Let $k \geq 1$. Let $\mathbf{a} = (a_1, \ldots, a_k)$ be a monotone $k$-tuple over $\mathbb{N}$. Then, there is exactly one $n \geq 0$ such that $\texttt{Generate}_k$ outputs $\mathbf{a}$ in iteration step $n$.*

**Proof:** We prove the lemma in three steps: first, we show that the generated tuples are monotone, second, we show that the generated tuples are ordered according to the defined order $<$, and third, we show that every monotone $k$-tuple over $\mathbb{N}$ is generated.

[$\texttt{Generate}_k$ outputs monotone $k$-tuples.]
We show the claim by induction over the iteration steps. In iteration step 0, $(0, \ldots, 0)$ is output, and this tuple is a monotone tuple. So, let $(a_1, \ldots, a_k)$ be the output in some iteration step $n$. By assumption, $(a_1, \ldots, a_k)$ is monotone, i.e., $a_1 \leq \cdots \leq a_k$. Let $j$ be the value of variable $i$ of $\texttt{Generate}_k$ in iteration step $n+1$. Hence, $a_1 = \cdots = a_j$ and $a_j < a_{j+1}$. Then, $a_j < a_j + 1 \leq a_{j+1} \leq \cdots \leq a_k$, and the output $(0, \ldots, 0, a_j + 1, a_{j+1}, \ldots, a_k)$ of iteration step $n+1$ is monotone.

[$\texttt{Generate}_k$ outputs $k$-tuples according to order $<$.]
Let $\mathbf{a} = (a_1, \ldots, a_k)$ be the output of $\texttt{Generate}_k$ in iteration step $n \geq 0$. By definition of $\texttt{Generate}_k$, there is $i \in \{1, \ldots, k\}$ such that $\mathbf{a}' = (0, \ldots, 0, a_i + 1, a_{i+1}, \ldots, a_k)$ is the output in iteration step $n+1$, where $\mathbf{a}'$ is monotone. Since $a_i < a_i + 1$, $\mathbf{a} < \mathbf{a}'$. By induction, it follows that the $k$-tuples output by $\texttt{Generate}_k$ are ordered according to $<$.

```
Generate$_k$:
begin
    $b_{k+1}$ := $\infty$;
    $b_1$ := 0; ...; $b_k$ := 0;
    output $(b_1, \ldots, b_k)$;
    for n := 1 to $\infty$ do
        $i$ := $\min\{i \in \{1, \ldots, k\} : b_i < b_{i+1}\}$;
        $b_i$ := $b_i$ + 1;
        $b_1$ := 0; ...; $b_{i-1}$ := 0;
        output $(b_1, \ldots, b_k)$
    end for
end.
```

**Figure 14** The entries of a monotone $k$-tuple over the set of naturals are ordered by $\leq$. Algorithm Generate$_k$ generates all monotone $k$-tuples over $\mathbb{N}$.

[$\texttt{Generate}_k$ outputs every monotone $k$-tuple.]
Since we have shown that the $k$-tuples output by $\texttt{Generate}_k$ are ordered according to $<$ and since there are only finitely many monotone $k$-tuples smaller than a given monotone $k$-tuple with respect to $<$, it suffices to show that, for every $n \geq 0$, if $\mathbf{a}_1$ is output in iteration step $n$ and $\mathbf{a}_2$ is output in iteration step $n+1$, there is no monotone $k$-tuple $\mathbf{a}$ such that $\mathbf{a}_1 < \mathbf{a} < \mathbf{a}_2$. Let $i \in \{1, \ldots, k\}$ such that $\mathbf{a}_1 = (a_1, \ldots, a_k)$ and $\mathbf{a}_2 = (0, \ldots, 0, a_i + 1, a_{i+1}, \ldots, a_k)$. If $\mathbf{a}$ such that $\mathbf{a}_1 < \mathbf{a} < \mathbf{a}_2$, then $i \geq 2$, $\mathbf{a} = (a'_1, \ldots, a'_{i-1}, a_i, \ldots, a_k)$ and there is $j \in \{1, \ldots, i-1\}$ such that $a_j < a'_j$ and $a_{j'} = a'_{j'}$ for all $j' \in \{j+1, \ldots, i-1\}$. By definition of $\texttt{Generate}_k$, $a_1 = \cdots = a_i$. But since $j < i$, $a'_j > a_i$, and $\mathbf{a}$ cannot be a monotone $k$-tuple.

■

From algorithm $\texttt{Generate}_k$, we easily obtain an algorithm that generates all $k$-tuples over $\mathbb{N}$. This algorithm is based on the following result, that defines a bijective mapping between the set of monotone $k$-tuples over $\mathbb{N}$ and the set of (arbitrary) $k$-tuples over $\mathbb{N}$.

**Lemma 7.2** *Let $k \geq 1$. The function that maps monotone $k$-tuples $(a_1, \ldots, a_k)$ to $(a_1, a_2 - a_1, \ldots, a_k - a_{k-1})$ defines a bijection between the sets of monotone $k$-tuples over $\mathbb{N}$ and $k$-tuples over $\mathbb{N}$.*

**Proof:** Let $f$ denote the mapping. Let $\mathbf{m} = (m_1, \ldots, m_k)$ be a $k$-tuple over $\mathbb{N}$. Then, $\mathbf{m}$ is the result of applying $f$ to $(m_1, m_1 + m_2, \ldots, m_1 + \cdots + m_k)$, which is a monotone $k$-tuple over $\mathbb{N}$. Hence, $f$ is surjective. Now, let $\mathbf{a}_1 = (a_1, \ldots, a_k)$ and $\mathbf{a}_2 = (a'_1, \ldots, a'_k)$ be such that $f(\mathbf{a}_1) = f(\mathbf{a}_2)$. By definition of $f$, $a_1 = a'_1$, and by induction, $a_i = a'_i$ for all $i \in \{1, \ldots, k\}$. Hence, $\mathbf{a}_1 = \mathbf{a}_2$, and $f$ is injective. Since $f$ is total, $f$ is bijective.

■

The original algorithm that we wish to obtain from $\texttt{Generate}_k$ realises the mapping defined in Lemma 7.2. However, it will not be a bijection. The algorithm, which is called $\texttt{Enumerate}_k$, is presented in Figure 15. We observe a number of differences between $\texttt{Generate}_k$ and $\texttt{Enumerate}_k$, but these are only slight modifications. Most important is it to see that the $k$-tuples over $\mathbb{N}$ that we wish to enumerate are contained in the variables $m_1, \ldots, m_k$ of $\texttt{Enumerate}_k$. To analyse $\texttt{Enumerate}_k$, we say that the value of a variable in some induction step is the value it contains at the end of the loop. Before we formally establish the connection between $\texttt{Generate}_k$ and $\texttt{Enumerate}_k$, consider the first few induction steps of $\texttt{Enumerate}_3$. Bold iteration step numbers mark iteration steps in which variable $i$ has value greater than 1. We will see that these iteration steps correspond to iteration steps of $\texttt{Generate}_k$.

```
Enumerate$_k$:
begin
    $b_k$ := 0;
    $m_1$ := 0; ...; $m_k$ := 0;
    $m_{k+1}$ := $\infty$;
    for n := 0 to $\infty$ do
        $i$ := $\min\{i \in \{1,\ldots,k{+}1\} : m_i \geq 1\}$;
        $m_i$ := $m_i$ - 1;
        $m_{i-1}$ := $b_{i-1}$;
        $b_{i-1}$ := $b_{i-1}$ + 1;
        $b_1$ := 1; ...; $b_{i-2}$ := 1;
        output $(m_1, \ldots, m_k)$
    end for
end.
```

**Figure 15** Based on the definition of `Generate`$_k$, `Enumerate`$_k$ generates all $k$-tuples over the set of naturals.

| n | **0** | **1** | **2** | **3** | 4 | **5** | **6** | **7** | 8 | **9** | **10** | 11 | **12** | 13 | 14 | **15** | **16** | **17** | 18 | **19** | **20** | 21 | **22** | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | 4 | 4 | 3 | 2 | 1 | 4 | 3 | 2 | 1 | 3 | 2 | 1 | 2 | 1 | 1 | 4 | 3 | 2 | 1 | 3 | 2 | 1 | 2 | 1 | 1 |
| $b_3$ | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $b_2$ | 1 | 1 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| $b_1$ | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 3 | 3 | 3 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 3 | 3 | 3 |
| $m_3$ | 0 | 1 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| $m_2$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 |
| $m_1$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 1 | 0 |

By a *phase* of $\texttt{Enumerate}_k$, we mean an iteration step in which $i$ has value greater than 1, and by phase $n$ of $\texttt{Enumerate}_k$, we mean the $n$th phase of $\texttt{Enumerate}_k$, where we start with $n = 0$.

**Lemma 7.3** *Let $k \geq 1$. Let $n \geq 0$.*

*(1) If $\texttt{Generate}_k$ outputs $(a_1, \ldots, a_k)$ in iteration step $n$, then $a_1 + 1, \ldots, a_k + 1$ are the values of the variables $b_1, \ldots, b_k$ in phase $n$ of $\texttt{Enumerate}_k$.*

*(2) If $a_1, \ldots, a_k$ are the values of the variables $b_1, \ldots, b_k$ in phase $n$ of $\texttt{Enumerate}_k$, respectively, $(a_1 - 1, a_2 - a_1, \ldots, a_k - a_{k-1})$ is output in phase $n$ of $\texttt{Enumerate}_k$.*

**Proof:** We show both statements simultaneously. Observe that, if, in some iteration step $n$ of $\texttt{Enumerate}_k$, $i$ has value 1, then $n \geq 1$ and the values of $b_1, \ldots, b_k$ in iteration step $n$ are equal to the values of $b_1, \ldots, b_k$ in iteration step $n-1$, respectively (lines 9 and 10 of $\texttt{Enumerate}_k$). A similar equality holds for variables $m_2, \ldots, m_k$ in iteration steps $n$ and $n - 1$ (lines 7 and 8 of $\texttt{Enumerate}_k$).

Let $n = 0$. By definition, $\texttt{Generate}_k$ outputs tuple $(0, \ldots, 0)$. By the definition of $\texttt{Enumerate}_k$, $i$ has value $k{+}1$ in iteration step 0, and $b_1 = \cdots = b_k = 1$ and $m_1 = \cdots = m_k = 0$. Hence, the statement holds for $n = 0$.

Let $n > 0$. Let $j$ be the value of variable $i$ in phase $n$ of $\texttt{Enumerate}_k$. By definition, $m_1 = \cdots = m_{j-1} = 0$ and $m_j > 0$ in phase $n-1$ of $\texttt{Enumerate}_k$. Applying the induction hypothesis, $b_1 = \cdots = b_{j-1} < b_j$ for the output in iteration step $n-1$ of $\texttt{Generate}_k$. Hence, $j$ is the value of variable $i$ of $\texttt{Generate}_k$ in interation step $n$, and statement (1) is true for $n$. For statement (2), it remains to see that $m_1 = b_1 - 1 = 0$ and $m_2 = \cdots = m_{j-1} = 0 = b_2 - b_1 = \cdots = b_{j-1} - b_{j-2}$ in phase $n$ of $\texttt{Enumerate}_k$. Furthermore, $b_j - b_{j-1} = m_j$ in phase $n$ of $\texttt{Enumerate}_k$. Since the values of $b_{j+1}, \ldots, b_k$ are equal in phases $n$ and $n - 1$ of $\texttt{Enumerate}_k$ as well as of $\texttt{Generate}_k$, the lemma follows.

■

Note that the output of every phase of algorithm $\texttt{Enumerate}_k$ is obtained from the values of the variables $b_1, \ldots, b_k$ of $\texttt{Enumerate}_k$ by application of the rule that was defined in Lemma 7.2.

**Corollary 7.4** *Let $k \geq 1$. Algorithm* $\texttt{Enumerate}_k$ *outputs every $k$-tuple over the set of natural numbers.*

**Proof:** Let $\mathbf{a}$ be a $k$-tuple over $\mathbb{N}$. Let $\mathbf{a}'$ be the monotone $k$-tuple that is mapped to $\mathbf{a}$ according to the definition of Lemma 7.2. By Lemma 7.1, $\mathbf{a}'$ is output by $\texttt{Generate}_k$, and due to Lemma 7.3, $\mathbf{a}$ is output by $\texttt{Enumerate}_k$ in some phase.

■

Remember that, if $\texttt{Enumerate}_k$ outputs only in phases, i.e., in iteration steps in which $i$ has value greater than 1, it outputs every $k$-tuple over $\mathbb{N}$ exactly once.

## 7.2 The description of a component

We have defined an algorithm that generates all $k$-tuples over $\mathbb{N}$ for $k \geq 1$. This algorithm uses only a few operations: assignments, conditional tests (finding the minimum) and addition and subtraction. We want to simulate the behaviour of this algorithm by recurrent systems. However, it takes considerable effort to simulate subtraction. In this section, we will define a recurrent system that can decrease a number by 1. To make the work easy, we will talk about another type of recurrent systems. Let $A, B \subseteq \mathbb{N}$. We define function gr, which should be understood as *greater than*, as follows: if $B \subseteq A \oplus \mathbb{N}$ then $\mathrm{gr}(A,B) =_{\mathrm{def}} \{0\}$; otherwise $\mathrm{gr}(A,B) =_{\mathrm{def}} \{1\}$. By 1, we (also) denote the constant 0-ary function 1.

**Definition 17** *Let $n \geq 1$. A finite recurrent system $S = (\mathcal{F}, A)$ over sets of naturals of dimension $n$ is a **finite recurrent** $\{\mathrm{gr}, \oplus, \otimes, 1\}$**-system** over sets of naturals of dimension $n$ if and only if every function in $\mathcal{F}$ is a $\{\mathrm{gr}, \oplus, \otimes, 1\}$-function.*

It is easy to see that recurrent $\{\mathrm{gr}, \oplus, \otimes, 1\}$-systems, as we will call finite recurrent $\{\mathrm{gr}, \oplus, \otimes, 1\}$-systems over sets of naturals for short henceforth, can generate only singleton sets in every iteration step. That is why we will not use set braces for singleton set inputs and results. For convenience, we define five auxiliary functions. Let $A, B \subseteq \mathbb{N}$. Then,

$$\begin{aligned}
\mathrm{ueq}(A,B) &=_{\mathrm{def}} \mathrm{gr}(A,B) \oplus \mathrm{gr}(B,A)\\
\mathrm{or}(A,B) &=_{\mathrm{def}} \mathrm{ueq}(A \oplus B, 0)\\
\mathrm{no}(A) &=_{\mathrm{def}} \mathrm{ueq}(A,1)\\
\mathrm{eq}(A,B) &=_{\mathrm{def}} \mathrm{no}(\mathrm{ueq}(A,B))\\
0 &=_{\mathrm{def}} \mathrm{no}(1)\,.
\end{aligned}$$

Note that ueq stands for *unequal*, no stands for the Boolean 'not', eq stands for *equal* and or stands for the Boolean 'or'. Furthermore, every function maps to either 0 or 1, which may be understood as Boolean 'false' and 'true'.

**Lemma 7.5** *Let $A, B \subseteq \mathbb{N}$.*

*(1)* $\mathrm{gr}(A,B) = 1$ *if and only if* $\min A > \min B$.

*(2)* $\mathrm{ueq}(A,B) = 0$ *or* $\mathrm{ueq}(A,B) = 1$,
*and* $\mathrm{ueq}(A,B) = 1$ *if and only if* $\min A \neq \min B$.

*(3)* $\mathrm{or}(A,B) = 1$ *if and only if* $0 \notin A$ *or* $0 \notin B$.

*(4)* $\mathrm{no}(A) = 1$ *if and only if* $\min A \neq 1$.

**Proof:** (1) If $\mathrm{gr}(A,B) = 1$, i.e., $B \nsubseteq A \oplus \mathbb{N}$, then $\min A > \min B$. If $\mathrm{gr}(A,B) = 0$, i.e., $B \subseteq A \oplus \mathbb{N}$, then $\min A \leq \min B$.

(2) Since $\mathrm{gr}(A,B) = 1$ implies $\mathrm{gr}(B,A) = 0$, $\mathrm{ueq}(A,B) = 0$ or $\mathrm{ueq}(A,B) = 1$. By symmetry, let us assume $\min A \leq \min B$. If $\min A = \min B$, then $\mathrm{gr}(A,B) = \mathrm{gr}(B,A) = 0$. If $\min A < \min B$, then $\mathrm{gr}(A,B) = 0$ and $\mathrm{gr}(B,A) = 1$, i.e., $\mathrm{ueq}(A,B) = 1$.

(3) If $0 \in A \cap B$, then $0 \in A \oplus B$, and $\min(A \oplus B) = \min\{0\}$. Otherwise, $\min(A \oplus B) > \min\{0\}$, and $\mathrm{ueq}(A \oplus B, 0) = 1$.

(4) Let $\min A = 1$. Then $A \subseteq 1 \oplus \mathbb{N}$ and $\{1\} \subseteq A \oplus \mathbb{N}$, or $\mathrm{gr}(1,A) = 0$ and $\mathrm{gr}(A,1) = 0$. This means that $\mathrm{ueq}(A,1) = \mathrm{no}(A) = 0$. For the converse, let $\min A \neq 1$. If $\min A = 0$, then $A \nsubseteq 1 \oplus \mathbb{N}$, if $\min A \geq 2$, then $\{1\} \nsubseteq A \oplus \mathbb{N}$. Hence, $\mathrm{ueq}(A,1) = \mathrm{no}(A) = 1$.

■

The recurrent $\{\mathrm{gr}, \oplus, \otimes, 1\}$-system, that we will define, will be $S_M = (\mathcal{F}_M, A_M)$ and will be of dimension 5. In the next section, many copies of this system will be composed to a new one, and these copies will depend on each other. This communication will be realised by functions. In the following system, these functions will already be present but without a precise definition. They are called $f_{a'}, f_{b'}, f_{c'}$ and have the following properties. Let $t \geq 0$.

$$\begin{aligned} f_{a'}(0) =_{\mathrm{def}} f_{b'}(0) =_{\mathrm{def}} f_{c'}(0) &=_{\mathrm{def}} 0 \\ f_{a'}(t+1) &=_{\mathrm{def}} f_{a'}(\mathcal{F}_M(t)) \\ f_{b'}(t+1) &=_{\mathrm{def}} f_{b'}(\mathcal{F}_M(t)) \\ f_{c'}(t+1) &=_{\mathrm{def}} f_{c'}(\mathcal{F}_M(t)) \end{aligned}$$

Furthermore, we use two auxiliary functions $f'$ and $f''$. Let $\mathbf{x} =_{\mathrm{def}} (a,b,c,h,m)$ and

$$\begin{aligned} f'(\mathbf{x}) &=_{\mathrm{def}} \mathrm{ueq}(h, 1 \oplus (m \otimes \mathrm{no}(f_{c'}(\mathbf{x})))) \\ f''(\mathbf{x}) &=_{\mathrm{def}} (m \otimes \mathrm{no}(f_{c'}(\mathbf{x}))) \oplus (f'(\mathbf{x}) \otimes \mathrm{ueq}(h, m \otimes \mathrm{no}(f_{c'}(\mathbf{x})))) \,. \end{aligned}$$

Now, let $\mathcal{F}_M =_{\text{def}} \langle f_a, f_b, f_c, f_h, f_m \rangle$ where

$$\begin{aligned}
f_a(\mathbf{x}) &=_{\text{def}} \text{eq}(h,m) \otimes \text{no}(f_{c'}(\mathbf{x}))\\
f_b(\mathbf{x}) &=_{\text{def}} (f_{a'}(\mathbf{x}) \otimes b) \oplus \text{or}(\text{no}(f_{a'}(\mathbf{x})), f_{c'}(\mathbf{x}))\\
f_c(\mathbf{x}) &=_{\text{def}} f_{c'}(\mathbf{x}) \otimes \text{eq}(h,0)\\
f_m(\mathbf{x}) &=_{\text{def}} f_{a'}(\mathbf{x}) \otimes f''(\mathbf{x})\\
f_h(\mathbf{x}) &=_{\text{def}} (f_{a'}(\mathbf{x}) \otimes ((f'(\mathbf{x}) \otimes h) \oplus (\text{no}(f'(\mathbf{x})) \otimes f''(\mathbf{x})))) \oplus (\text{no}(f_{a'}(\mathbf{x})) \otimes f_{b'}(\mathbf{x}))\,.
\end{aligned}$$

Finally, let $A_M =_{\text{def}} (0,0,0,0,0)$. We will continue by proving important properties of the behaviour of $S_M$. The proofs are very technical. Remember that no $\{\text{gr}, \oplus, \otimes, 1\}$-function on only non-empty input sets can output the empty set.

**Lemma 7.6** *(1) If $S_M[f_b](1) \subseteq \mathbb{N} \oplus 1$, then $S_M[f_b](t) \subseteq \mathbb{N} \oplus 1$ for every $t \geq 1$.*

*(2) $S_M[f_h](t) \subseteq S_M[f_m](t) \oplus \mathbb{N}$ for every $t \geq 0$.*

**Proof:** We prove the claims by induction over $t$. For statement (1), let $S_M[f_b](t) \subseteq \mathbb{N} \oplus 1$. If $0 \notin f_{a'}(t+1)$, then $f_{a'}(t+1) \otimes S_M[f_b](t) \subseteq \mathbb{N} \oplus 1$. If $0 \in f_{a'}(t+1)$, then $\text{no}(f_{a'}(t+1)) = 1$, and $S_M[f_b](t+1) \subseteq \mathbb{N} \oplus 1$.

For statement (2), note that $S_M[f_m](0) = S_M[f_h](0) = 0$, since $f_{a'}(0) = f_{b'}(0) = 0$. For the induction step, let $S_M[f_h](t) \subseteq S_M[f_m](t) \oplus \mathbb{N}$. First, consider the case $0 \in f_{a'}(t+1)$. Then, $0 \in S_M[f_m](t+1)$, and $S_M[f_m](t+1) \oplus \mathbb{N} = \mathbb{N}$. Now, let $0 \notin f_{a'}(t+1)$. If $\text{no}(f'(\mathcal{F}_M(t))) = 1$, the claim obviously holds. If $\text{no}(f'(\mathcal{F}_M(t))) = 0$, i.e., $f'(\mathcal{F}_M(t)) = 1$ by definition of $f'$, then $S_M[f_h](t+1) \subseteq f_{a'}(t+1) \otimes S_M[f_h](t)$. If $\text{no}(f_{c'}(t+1)) = 0$, then $0 = f''(\mathcal{F}_M(t))$ or $1 = f''(\mathcal{F}_M(t))$, and the claim holds. So, assume $\text{no}(f_{c'}(t+1)) = 1$. If $\text{ueq}(S_M[f_h](t), S_M[f_m](t)) = 0$, then $f''(\mathcal{F}_M(t)) = S_M[f_m](t)$, and the claim holds due to induction hypothesis. So, finally, let $\text{ueq}(S_M[f_h](t), S_M[f_m](t)) = 1$, i.e., $\min S_M[f_h](t) \neq \min S_M[f_m](t)$, and $f''(\mathcal{F}_M(t)) = S_M[f_m](t) \oplus 1$. By induction hypothesis, $S_M[f_h](t) \subseteq S_M[f_m](t) \oplus \mathbb{N}$, i.e., $\min S_M[f_h](t) \geq \min S_M[f_m](t)$, so that we obtain $\min S_M[f_h](t) > \min S_M[f_m](t)$, and the claim holds.

■

**Lemma 7.7** *Let $0 \leq t_1 \leq t_2$.*

*(1) Let $S_M[f_a](t_1) = f_{a'}(t_1) = 1$ and $f_{c'}(t_1) = 0$.
If $f_{a'}(t_1{+}1) = 1$ and $f_{c'}(t_1{+}1) = 0$, then $S_M[f_a](t_1{+}1) = 1$, $S_M[f_c](t_1{+}1) = 0$, $S_M[f_b](t_1{+}1) = S_M[f_b](t_1)$ and $S_M[f_m](t_1{+}1) = S_M[f_m](t_1)$.*

*(2) If $f_{a'}(t) = 0$ and $f_{b'}(t) \subseteq \mathbb{N} \oplus 1$ for all $t \in \{t_1, \ldots, t_2\}$ and $f_{c'}(t_1) = 1$, then $S_M[f_a](t) = 0$ for all $t \in \{t_1, \ldots, t_2\}$.*

**Proof:** Consider the assumptions of the first claim. Due to definition, $f_{a'}(0) = 0$, hence $t_1 > 0$. Furthermore, note that $\text{no}(f_{c'}(t_1)) = 1$. By definition of $f_a$, $S_M[f_a](t_1) = 1$ implies $S_M[f_h](t_1 - 1) = S_M[f_m](t_1 - 1)$, so that

$$S_M[f_m](t_1) = f''(\mathcal{F}_M(t_1 - 1)) = S_M[f_m](t_1 - 1)\,.$$

Since

$$S_M[f_h](t_1) = \begin{cases} S_M[f_h](t_1-1) & \text{, if } f'(\mathcal{F}_M(t_1-1)) = 1 \\ f''(\mathcal{F}_M(t_1-1)) & \text{, if } f'(\mathcal{F}_M(t_1-1)) = 0 \end{cases}$$

it follows that $S_M[f_h](t_1) = S_M[f_m](t_1)$, which results in $S_M[f_a](t_1+1) = 1$ and $\text{ueq}(S_M[f_h](t_1), S_M[f_m](t_1)) = 0$. By assumption, $f_{a'}(t_1+1) = 1$, so that

$$S_M[f_m](t_1+1) = f''(\mathcal{F}_M(t_1)) = S_M[f_m](t_1)\,.$$

The claims for $S_M[f_c](t_1+1)$ and $S_M[f_b](t_1+1)$ follow trivially by definition.

For proving the second claim, observe that $f_{c'}(t_1) = 1$ implies $S_M[f_a](t_1) = 0$. Furthermore, since $f_{a'}(t) = 0$ and $f_{b'}(t) \subseteq \mathbb{N} \oplus 1$ by assumption, $S_M[f_m](t) = 0$ and $S_M[f_h](t) \subseteq \mathbb{N} \oplus 1$ by definition, which yields $\text{eq}(f_m(t), f_h(t)) = 0$ and therefore $S_M[f_a](t) = 0$ for $t_1 < t \leq t_2$.

■

**Lemma 7.8** *Assume, for all* $t \geq 0$, $f_{c'}(t{+}1) = 1$ *implies* $S_M[f_a](t) = 1$, *and if* $f_{a'}(t) = 1$ *and* $f_{a'}(t{+}1) = 0$, *then* $f_{c'}(t{+}1) = 1$.
*Let* $t_0 \geq 0$ *be such that* $S_M[f_a](t_0) = 0$ *and* $f_{a'}(t_0) = 1$. *There is* $t_1 > t_0$ *such that* $S_M[f_a](t_1) = 1$. *If* $t_1$ *is smallest possible and* $\text{eq}(S_M[f_h](t_0), S_M[f_m](t_0)) = 0$ *then* $\text{eq}(S_M[f_h](t_0), (S_M[f_m](t_1) \oplus 1)) = 1$.

**Proof:** Let $k \geq 0$ be the value such that

$$\text{eq}(S_M[f_h](t_0), (S_M[f_m](t_0) \oplus \{k\})) = 1\,.$$

The existence of $k$ is due to Lemma 7.6. We show the claim by induction over $k$.

Let $k = 0$. It holds that $S_M[f_a](t_0+1) = 0$ implies $f_{c'}(t_0+1) = 1$ by definition of $f_a$ and $S_M[f_a](t_0) = 1$ by assumption. So, since $S_M[f_a](t_0) = 0$ is assumed, $S_M[f_a](t_0+1) = 1$, and $t_1 =_{\text{def}} t_0 + 1$ fulfills the claim. Furthermore, $t_1$ is smallest possible with this property and $S_M[f_m](t_1) = S_M[f_m](t_1 - 1)$.

Let $k \geq 1$. Clearly $S_M[f_a](t_0+1) = 0$, and $f_{a'}(t_0+1) = 1$ and $f_{c'}(t_0+1) = 0$ due to our assumptions. We distinguish two cases.

Let $k = 1$. It holds that

$$f'(\mathcal{F}_M(t_0+1)) = \text{ueq}(S_M[f_h](t_0+1), (1 \oplus S_M[f_m](t_0+1))) = 0$$

and

$$S_M[f_h](t_0+1) = S_M[f_m](t_0+1) = f''(\mathcal{F}_M(t_0+1)) = S_M[f_m](t_0)\,.$$

Now, $\text{eq}(S_M[f_h](t_0+1), S_M[f_m](t_0+1)) = 1$, and the existence of $t_1 > t_0$ such that $S_M[f_a](t_1) = 1$ follows by applying case $k = 0$. Furthermore, $t_1 = t_0 + 2$ and $S_M[f_m](t_1) = S_M[f_m](t_1 - 1) = S_M[f_m](t_0)$. Hence,

$$\text{eq}(S_M[f_h](t_0), (S_M[f_m](t_1) \oplus 1)) = 1\,.$$

Let $k \geq 2$. We conclude that $f'(\mathcal{F}_M(t_0)) = 1$, and

$$S_M[f_m](t_0+1) = f''(\mathcal{F}_M(t_0)) = S_M[f_m](t_0) \oplus 1$$

and $S_M[f_h](t_0+1) = S_M[f_h](t_0)$. It holds that the number $k'$ such that

$$\mathrm{eq}(S_M[f_h](t_0+1), (S_M[f_m](t_0+1) \oplus \{k'\})) = 1$$

is one less than $k$, and we can conclude the proof by applying the induction hypothesis.

■

## 7.3 Assembling components

In this section, we will finally obtain the recurrent $\{\mathrm{gr}, \oplus, \otimes, 1\}$-system that simulates $\texttt{Enumerate}_k$. We will assemble copies of system $S_M$. Let $S_M^{(1)}, \ldots, S_M^{(k)}$ be $k$ copies of system $S_M$. By $f_a^{(i)}, f_{a'}^{(i)}, \ldots, f_h^{(i)}, f_m^{(i)}$ we denote the functions of $S_M^{(i)}$, $i \in \{1, \ldots, k\}$. Consider the following definition.

$$\begin{aligned}
\mathcal{F}_E^k &=_{\mathrm{def}} \langle f_a^{(1)}, f_b^{(1)}, f_c^{(1)}, f_h^{(1)}, f_m^{(1)}, f_a^{(2)}, \ldots, f_b^{(k)}, f_c^{(k)}, f_h^{(k)}, f_m^{(k)}, f_A\rangle \\
\mathbf{x} &=_{\mathrm{def}} (a_1,\ b_1,\ c_1,\ h_1,\ m_1,\ a_2,\ \ldots,\ b_k,\ c_k,\ h_k,\ m_k,\ x)\,.
\end{aligned}$$

Furthermore, let $f_a^{(i)}(\mathbf{x}) = f_a(a_i, b_i, c_i, h_i, m_i)$, and similarly for $f_b^{(i)}, f_c^{(i)}, f_h^{(i)}, f_m^{(i)}$, $i \in \{1, \ldots, k\}$. Finally, let $f_A(\mathbf{x}) =_{\mathrm{def}} x \oplus f_c^{(k)}(\mathbf{x})$. Now, we can precise the definitions of $f_{a'}^{(i)}, f_{b'}^{(i)}, f_{c'}^{(i)}$, that appear in the components $S_M^{(1)}, \ldots, S_M^{(k)}$. For every $i \in \{1, \ldots, k-1\}$, let

$$\begin{aligned}
f_{a'}^{(i)}(\mathbf{x}) &=_{\mathrm{def}} f_a^{(i+1)}(\mathbf{x}) & & & f_{b'}^{(i)}(\mathbf{x}) &=_{\mathrm{def}} f_b^{(i+1)}(\mathbf{x}) & & & f_{c'}^{(1)}(\mathbf{x}) &=_{\mathrm{def}} a_1 \\
& & \text{and} & & & & \text{and} & & & \\
f_{a'}^{(k)}(\mathbf{x}) &=_{\mathrm{def}} \mathrm{no}(f_c^{(k)}(\mathbf{x})) & & & f_{b'}^{(k)}(\mathbf{x}) &=_{\mathrm{def}} f_A(\mathbf{x}) & & & f_{c'}^{(i+1)}(\mathbf{x}) &=_{\mathrm{def}} f_c^{(i)}(\mathbf{x})\,.
\end{aligned}$$

Let $S_E^k =_{\mathrm{def}} (\mathcal{F}_E^k, A_E^k)$ where $A_E^k =_{\mathrm{def}} (1, 0, 0, \ldots, 0)$. Observe that all functions are well-defined in the sense that no value depends on itself. We begin by showing some technical properties of $S_E^k$.

**Lemma 7.9** *Let $k \geq 1$.*

*(1)* $S_E^k[f_a^{(1)}](1) = \cdots = S_E^k[f_a^{(k)}](1) = 0\,.$

*(2)* $S_E^k[f_c^{(1)}](1) = \cdots = S_E^k[f_c^{(k)}](1) = 1\,.$

*(3)* $S_E^k[f_b^{(1)}](t), \ldots, S_E^k[f_b^{(k)}](t) \subseteq \mathbb{N} \oplus 1$ *for every* $t \geq 1\,.$

**Proof:** We prove the lemma by induction over $i$. We begin with functions $S_E^k[f_c^{(i)}]$. Observe that $S_E^k[f_h^{(i)}](0) = 0$ for all $i \in \{1, \ldots, k\}$, and therefore, $\mathrm{eq}(S_E^k[f_h^{(i)}](0), 0) = 1$. By definition, $f_{c'}^{(1)}(1) = 1$, hence, $S_E^k[f_c^{(1)}](1) = 1$. Now, let $S_E^k[f_c^{(i)}](1) = 1$, for

$1 \leq i < k$. Since $S_E^k[f_c^{(i+1)}](1) = f_{c'}^{(i+1)}(1) = S_E^k[f_c^{(i)}](1) = 1$, the lemma holds for $S_E^k[f_c^{(i)}]$. Note that we have also shown $f_{c'}^{(i)}(1) = 1$. It immediately follows the results for functions $S_E^k[f_a^{(i)}]$ and $S_E^k[f_b^{(i)}]$ by definitions.

■

Note that $S_E^k[f_a^{(i)}](t), f_{a'}^{(i)}(t), S_E^k[f_c^{(i)}](t), f_{c'}^{(i)}(t) \in \{0,1\}$ for every $i \in \{1,\ldots,k\}$ and $t \geq 0$. Furthermore, every function of $\mathcal{F}_E^k$ always computes singleton sets.

**Lemma 7.10** *For every $t \geq 0$ and $i \in \{1,\ldots,k\}$,*

*(1) if $S_E^k[f_a^{(i)}](t) = 1$ and $S_E^k[f_a^{(i)}](t+1) = 0$ then $f_{c'}^{(i)}(t+1) = 1$*

*(2) if $f_{a'}^{(i)}(t) = 0$ then $S_E^k[f_a^{(i)}](t) = 0$*

*(3) $f_{c'}^{(1)}(t+1) = 1$ if and only if $S_E^k[f_a^{(1)}](t) = \cdots = S_E^k[f_a^{(k)}](t) = 1$.*

**Proof:** Since $f_{c'}^{(1)}(t+1) = S_E^k[f_a^{(1)}](t)$ by definition, statement (1) holds for $i = 1$. We show the statement for $i > 1$ by induction over $i$. Assume that $S_E^k[f_a^{(i)}](t) = 1$ and $S_E^k[f_a^{(i)}](t+1) = 0$. It holds that $t \geq 1$ and $f_{c'}^{(i)}(t) = 0$. In particular, $S_E^k[f_c^{(i)}](t) = 0$.

Suppose $f_{c'}^{(i)}(t+1) = 0$. Then, $\text{eq}(S_E^k[f_h^{(i)}](t), S_E^k[f_m^{(i)}](t)) = 0$, i.e., $S_E^k[f_h^{(i)}](t) \neq S_E^k[f_m^{(i)}](t)$. Due to $S_E^k[f_a^{(i)}](t) = 1$, $S_E^k[f_h^{(i)}](t-1) = S_E^k[f_m^{(i)}](t-1)$, and $f_{a'}^{(i)}(t-1) = 1$. We will show that $f_{a'}^{(i)}(t) = 0$ must hold. Suppose $f_{a'}^{(i)}(t) = 1$. Then, $S_E^k[f_h^{(i)}](t) = S_E^k[f_h^{(i)}](t-1)$. If $\text{no}(f_{c'}^{(i)}(t)) = 1$, then $S_E^k[f_m^{(i)}](t) = S_E^k[f_m^{(i)}](t-1)$, which contradicts the assumption. Hence, $\text{no}(f_{c'}^{(i)}(t)) = 0$, which contradicts $S_E^k[f_a^{(i)}](t) = 1$. So, $f_{a'}^{(i)}(t) = 0$. If $i = k$, $S_E^k[f_c^{(k)}](t) = 1$ by definition, which contradicts the assumption. Let $1 < i < k$. By definition of $S_E^k$, we obtain $S_E^k[f_a^{(i+1)}](t-1) = 1$ and $S_E^k[f_a^{(i+1)}](t) = 0$. Applying the induction hypothesis, $f_{c'}^{(i+1)}(t) = 1$, and by definition of $S_E^k$, $S_E^k[f_c^{(i)}](t) = 1$, which is a contradiction. Hence, $f_{c'}^{(i)}(t+1) = 1$.

We prove statement (2). Observe that $f_{c'}^{(i)}(t) = 1$ implies $S_E^k[f_a^{(i)}](t) = 0$ by definition. Hence, it suffices to show that, if $f_{a'}^{(i)}(t) = 0$, then $f_{c'}^{(i)}(t) = 1$. We show this claim by induction over $i$. Let $i = k$. Since $f_{a'}^{(k)}(t) = \text{no}(S_E^k[f_c^{(k)}](t))$, we obtain $S_E^k[f_c^{(k)}](t) = 1$. In particular, $f_{c'}^{(k)}(t) = 1$. Now, let $i < k$. By definition, $S_E^k[f_a^{(i+1)}](t) = 0$, and by induction hypothesis, $f_{c'}^{(i+1)}(t) = S_E^k[f_c^{(i)}](t) = f_{c'}^{(i)}(t) = 1$, which proves statement (2).

For statement (3), observe that $S_E^k[f_a^{(1)}](t) = 1$ immediately implies $f_{c'}^{(1)}(t+1)$. So, for the converse, let $f_{c'}^{(1)}(t+1) = 1$, i.e., $S_E^k[f_a^{(1)}](t) = 1$. By induction, we obtain the result from statement (2).

■

**Lemma 7.11** *Let $t_0 \geq 0$ such that $f_{c'}^{(1)}(t_0) = 1$. Then, there is $t_1 > t_0$ such that $S_E^k[f_a^{(1)}](t_1) = 1$.*

**Proof:** Observe that, if $f_{c'}^{(i)}(t_0) = 1$, then $S_E^k[f_a^{(i)}](t_0) = 0$. Let $i \in \{1, \ldots, k\}$ be largest possible such that $S_E^k[f_a^{(i)}](t_0) = 0$. We will show that there is $t_1 > t_0$ such that $S_E^k[f_a^{(k)}](t_1) = \cdots = S_E^k[f_a^{(i)}](t_1) = 1$. Iterated application proves the statement.

Note that the assumptions of Lemma 7.8 are fulfilled for component $i$ of $S_M$. For the first assumption, note that $f_{c'}^{(i)}(t+1) = 1$ implies $f_{c'}^{(1)}(t+1) = 1$, and due to Lemma 7.10, $S_E^k[f_a^{(i)}](t) = 1$. For the second assumption, if $i = k$, $f_{a'}^{(k)}(t+1) = 0$, then $S_E^k[f_c^{(k)}](t+1) = 1$ and $f_{c'}^{(k)}(t+1) = 1$, if $i < k$, $f_{a'}^{(i)}(t) = S_E^k[f_a^{(i+1)}](t) = 1$ and $f_{a'}^{(i)}(t+1) = S_E^k[f_a^{(i+1)}](t+1) = 0$, and due to Lemma 7.10, $f_{c'}^{(i+1)}(t+1) = S_E^k[f_c^{(i)}](t+1) = f_{c'}^{(i)}(t+1) = 1$.

Let $i = k$. Due to Lemma 7.10, $f_{c'}^{(1)}(t_0+1) = f_{c'}^{(k)}(t_0+1) = S_E^k[f_c^{(k)}](t_0+1) = 0$. Since $f_{a'}^{(k)}(t_0+1) = \text{no}(S_E^k[f_c^{(k)}](t_0+1))$ by definition, $f_{a'}^{(k)}(t_0+1) = 1$. If $S_E^k[f_a^{(k)}](t_0+1) = 0$, we can apply Lemma 7.8 and obtain $t_1 > t_0+1$ such that $S_E^k[f_a^{(k)}](t_1) = 1$. Let $i < k$. By definition, $f_{a'}^{(i)}(t_0) = S_E^k[f_a^{(i+1)}](t_0)$, and due to the maximality of $i$, $S_E^k[f_a^{(i+1)}](t_0) = 1$. We apply Lemma 7.8 and obtain $t_1$ such that $S_E^k[f_a^{(i)}](t_1) = 1$. Applying statement (2) of Lemma 7.10, we obtain that $S_E^k[f_a^{(k)}](t_1) = \cdots = S_E^k[f_a^{(i)}](t_1) = 1$, and we conclude the proof.

■

Statements (2) and (3) of Lemma 7.10 and Lemma 7.11 justify the following definition of a phase of $S_E^k$. A *phase* of $S_E^k$ is an interval $[t_0, t_1]$ for numbers $t_0$ and $t_1$, $t_0 < t_1$, where $f_{c'}^{(1)}(t_0) = 1$, $S_E^k[f_a^{(1)}](t_1) = 1$ and $f_{c'}^{(1)}(t) = 0$ for every $t \in \{t_0+1, \ldots, t_1\}$. In other words, $t_1$ is the smallest value such that $t_1 > t_0$ and $S_E^k[f_a^{(1)}](t_1) = 1$. Note that $S_E^k[f_a^{(1)}](t_0) = 0$. Furthermore, there is no $t_1$ such that $[0, t_1]$ is a phase of $S_E^k$. The following lemma shows the central result about the components of $S_E^k$, in fact, that they perform subtraction.

**Lemma 7.12** *Let $t_0$ and $t_1$, $0 < t_0 < t_1$, be such that $[t_0, t_1]$ is a phase of $S_E^k$, and let $i \in \{1, \ldots, k\}$ such that $f_{c'}^{(i)}(t_0) = 1$. If $S_E^k[f_c^{(i)}](t_0) = 0$, then $S_E^k[f_m^{(i)}](t_0 - 1) = S_E^k[f_m^{(i)}](t_1) \oplus 1$. Otherwise $f_{b'}^{(i)}(t_1) = S_E^k[f_m^{(i)}](t_1) \oplus 1$.*

**Proof:** Due to definition, $S_E^k[f_a^{(i)}](t_0) = 0$. By the definition of a phase, there is a smallest number $t'$ where $t_0 \le t' \le t_1$ such that $f_{a'}^{(i)}(t') = 1$. Note that $S_E^k[f_a^{(i)}](t') = 0$ (statement (2) of Lemma 7.10 and the definition of $f_a^{(i)}$).

Let $S_E^k[f_c^{(i)}](t_0) = 0$. Then, $t' = t_0$ due to statement (1) of Lemma 7.10, and $S_E^k[f_h^{(i)}](t_0 - 1) \neq 0$. Since $S_E^k[f_a^{(i)}](t_0 - 1) = 1$, it holds that $S_E^k[f_h^{(i)}](t_0 - 2) = S_E^k[f_m^{(i)}](t_0 - 2)$. Also, $f_{a'}^{(i)}(t_0 - 1) = 1$ due to statement (2) of Lemma 7.10, and $f_{c'}^{(i)}(t_0 - 1) = 0$. Thus, $S_E^k[f_h^{(i)}](t_0 - 1) = S_E^k[f_h^{(i)}](t_0 - 2)$ and $S_E^k[f_m^{(i)}](t_0 - 1) = S_E^k[f_m^{(i)}](t_0 - 2)$. If $S_E^k[f_h^{(i)}](t_0 - 1) = 1$, then $S_E^k[f_h^{(i)}](t_0) = S_E^k[f_m^{(i)}](t_0) = 0$, and $S_E^k[f_a^{(i)}](t_0 + 1) = 1$. So, let $S_E^k[f_h^{(i)}](t_0 - 1) \neq 1$. Then, $S_E^k[f_h^{(i)}](t_0) = S_E^k[f_h^{(i)}](t_0 - 1)$

and $S^k_E[f^{(i)}_h](t_0) \neq S^k_E[f^{(i)}_m](t_0)$, and we can apply Lemma 7.8. Hence, there is $t_1 > t_0$ such that $S^k_E[f^{(i)}_h](t_0) = S^k_E[f^{(i)}_m](t_1) \oplus 1$. By the results of Lemma 7.7, we conclude this case.

Now, let $S^k_E[f^{(i)}_c](t_0) = 1$. Then, $t' > t_0$. Note that $S^k_E[f^{(i)}_h](t'-1) = f^{(i)}_{b'}(t'-1)$ and $S^k_E[f^{(i)}_m](t'-1) = 0$. By the choice of $t'$, $f^{(k)}_{b'}(t') = f^{(k)}_{b'}(t'-1)$ and, since $f^{(i)}_{a'}(t') = 1$ and $f^{(i)}_{c'}(t') = 0$, $f^{(i)}_{b'}(t') = f^{(i)}_{b'}(t'-1)$ for $i < k$. Hence, $f^{(i)}_{b'}(t_1) = f^{(i)}_{b'}(t'-1)$. So, the situation of component $i$ in $t'$ equals the situation in $t_0$ of the first case, and we can conclude the proof.

■

**Corollary 7.13** *Let $[t_0, t_1]$ be a phase of $S^k_E$, and let $i \in \{1, \dots, k\}$ such that $S^k_E[f^{(i)}_c](t_0) = 1$. Then, $S^k_E[f^{(i)}_b](t_1) = \cdots = S^k_E[f^{(1)}_b](t_1) = 1$ and $S^k_E[f^{(i-1)}_m](t_1) = \cdots = S^k_E[f^{(1)}_m](t_1) = 0$.*

**Proof:** Let $t \in \{t_0, \dots, t_1\}$ be largest possible such that $f^{(i)}_{a'}(t) = 0$. It is clear that $S^k_E[f^{(i)}_b](t) = 1$. Since $f^{(i)}_{a'}(t') = 1$ and $f^{(i)}_{c'}(t) = 0$ for every $t \in \{t+1, \dots, t_1\}$, $S^k_E[f^{(i)}_b](t_1) = 1$. By iterated application of these arguments, we obtain $S^k_E[f^{(i)}_b](t_1) = \cdots = S^k_M[f^{(1)}_b](t_1) = 1$, and the claim follows from Lemma 7.12.

■

We can order the phases of $S^k_E$ according to the left or right endpoints and enumerate them accordingly starting with phase 0. Let $s^{(n)}$ denote the right endpoint of phase $n$, i.e., there is a number $r$ such that $[r, s^{(n)}]$ is phase $n$. Note that $s^{(n)} + 1$ is the left endpoint of phase $n+1$. Then, for every $n \geq 0$, let

$$\begin{aligned}\mu(n) &=_{\text{def}} (S^k_E[f^{(k)}_m](s^{(n)}), \dots, S^k_E[f^{(1)}_m](s^{(n)}))\\ \kappa(n) &=_{\text{def}} (f^{(k)}_{b'}(s^{(n)}), \dots, f^{(1)}_{b'}(s^{(n)}))\,.\end{aligned}$$

Since the entries of $\mu(n)$ are only singleton sets, we can say that $\mu(n)$ represents an element of $\mathbb{N}^k$. We will show that the sequences defined by $\mu$ and generated by $\texttt{Enumerate}_k$ can be considered equal. More precisely, the phases of $S^k_E$ exactly correspond to the iteration steps of $\texttt{Enumerate}_k$. For every $n \geq 0$, let

$$\begin{aligned}\nu(n) &=_{\text{def}} (m^{(n)}_k, \dots, m^{(n)}_1)\\ \beta(n) &=_{\text{def}} (b^{(n)}_k, \dots, b^{(n)}_1)\end{aligned}$$

where $m^{(n)}_j$ and $b^{(n)}_j$ denote the values of variables $m_j$ and $b_j$ in iteration step $n$ of $\texttt{Enumerate}_k$, respectively. For being mathematically precise, consider the following definition. Let $a_1, \dots, a_k, a'_1, \dots, a'_k \in \mathbb{N}$. Then, $(a_1, \dots, a_k) \simeq (\{a'_1\}, \dots, \{a'_k\})$ if and only if $a_i = a'_i$ for all $i \in \{1, \dots, k\}$.

**Proposition 7.14** *For every $n \geq 0$: $\mu(n) \simeq \nu(n)$.*

**Proof:** We will show the claim by induction over the iteration steps of $\texttt{Enumerate}_k$ and the phases of $S_E^k$. To achieve this, we will show a stronger result, in fact, that, for every $n \geq 0$, $\mu(n) \simeq \nu(n)$ and $\kappa(n) \simeq \beta(n)$. Subsequently, $i$ means the variable of $\texttt{Enumerate}_k$.

By definition of $\texttt{Enumerate}_k$, $\nu(0) = (0, \dots, 0)$ and $\beta(0) = (1, \dots, 1)$. For $S_E^k$, phase 0 has left endpoint 1. Due to Lemma 7.9, $S_E^k[f_c^{(k)}](1) = 1$, so that, due to Corollary 7.13, $\mu(0) = (S_E^k[f_m^{(k)}](s^{(0)}), 0, \dots, 0)$ and $\kappa(0) = (S_E^k[f_A](s^{(0)}), 1, \dots, 1)$. By construction and due to Lemma 7.10, it holds that $S_E^k[f_A](1) = S_E^k[f_A](s^{(0)}) = 1$. Finally, $S_E^k[f_m^{(k)}](s^{(0)}) = 0$ by Lemma 7.12. Hence, the claim holds for $n = 0$.

Now, let $n \geq 0$, and let $\mu(n) \simeq \nu(n)$ and $\kappa(n) \simeq \beta(n)$. Let $r \in \{1, \dots, k\}$ be largest possible such that $f_{c'}^{(r)}(s^{(n)}+1) = 1$. Recall that $s^{(n)}+1$ is the left endpoint of phase $n+1$ and that $S_E^k[f_h^{(j)}](s^{(n)}) = S_E^k[f_m^{(j)}](s^{(n)})$ for every $j \in \{1, \dots, k\}$ due to the definition of function $f_a^{(j)}$ and the end of a phase of $S_E^k$. If $S_E^k[f_c^{(k)}](s^{(n)} + 1) = 1$ then $\mu(n) = (0, \dots, 0)$, and $i = k + 1$ in $\texttt{Enumerate}_k$ in iteration step $n+1$. Hence, $\nu(n+1) = (b_k^{(n)}, 0, \dots, 0)$ and $\beta(n+1) = (b_k^{(n)} + 1, 1, \dots, 1)$. Due to definition of $S_E^k$ and since $f_{a'}^{(k)}(t) = 1$ for every $t \in \{s^{(n)} + 2, \dots, s^{(n+1)}\}$,

$$f_{b'}^{(k)}(s^{(n+1)}) = f_{b'}^{(k)}(s^{(n)} + 1) = f_{b'}^{(k)}(s^{(n)}) \oplus 1\,.$$

From Corollary 7.13, we obtain

$$\begin{aligned}\mu(n+1) =&(S_E^k[f_m^{(k)}](s^{(n+1)}), 0, \dots, 0)\\ \kappa(n+1) =&(f_{b'}^{(k)}(s^{(n)}) \oplus 1, 1, \dots, 1)\,,\end{aligned}$$

and Lemma 7.12 yields $S_E^k[f_m^{(k)}] = f_{b'}^{(k)}(s^{(n)})$, i.e., $\mu(n+1) = (f_{b'}^{(k)}(s^{(n)}), 0, \dots, 0)$, so that the claim holds in this case.

Now, let $S_E^k[f_c^{(k)}](s^{(n)} + 1) = 0$. So, $1 \leq r \leq k$, $r$ is the least number for which holds $S_E^k[f_m^{(r)}](s^{(n)}) \neq 0$, and $r$ is equal to the value of $i$ in $\texttt{Enumerate}_k$ in iteration step $n+1$. By definition of $\texttt{Enumerate}_k$,

$$\begin{aligned}\nu(n+1) =&(m_k^{(n)}, \dots, m_{r+1}^{(n)}, m_r^{(n)} - 1, \quad b_{r-1}^{(n)} \quad, 0, \dots, 0)\\ \beta(n+1) =&(\ b_k^{(n)}, \quad \dots \quad, \quad b_r^{(n)} \quad, b_{r-1}^{(n)} + 1, 1, \dots, 1)\,.\end{aligned}$$

For $S_E^k$, note that $f_{a'}^{(r)}(t) = 1$ for all $t \in \{s^{(n)} + 1, \dots, s^{(n+1)}\}$ and $f_{c'}^{(r)}(t) = 0$ for all $t \in \{s^{(n)} + 2, \dots, s^{(n+1)}\}$. Hence, by definition of $S_M$,

$$S_E^k[f_b^{(r)}](s^{(n+1)}) = S_E^k[f_b^{(r)}](s^{(n)} + 1) = S_E^k[f_b^{(r)}](s^{(n)}) \oplus 1\,.$$

Furthermore, $S_E^k[f_m^{(r)}](s^{(n)}) = S_E^k[f_m^{(r)}](s^{(n+1)}) \oplus 1$ and, if $r > 1$, $S_E^k[f_b^{(r)}](s^{(n+1)}) = S_E^k[f_m^{(r-1)}] \oplus 1$ due to Lemma 7.12. Applying Lemma 7.7 and Corollary 7.13, we finally obtain $\mu(n+1) \simeq \nu(n+1)$ and $\kappa(n+1) \simeq \beta(n+1)$.

■

**Corollary 7.15** *Let $k \geq 1$. For every $\mathbf{a} \in \mathbb{N}^k$ there is $n \in \mathbb{N}$ such that $\mathbf{a} \simeq \mu(n)$.*

**Proof:** Let $\mathbf{a}$ be a $k$-tuple over $\mathbb{N}$. Due to Corollary 7.4, there is an iteration step $n$ such that $\texttt{Enumerate}_k$ outputs tuple $\mathbf{a}$ in iteration step $n$. By Proposition 7.14, $\mathbf{a} \simeq \mu(n)$.

■

## 7.4 Undecidability results

There are two finishing steps that remain. First, we will show that the existential membership problem for recurrent $\{\text{gr}, \oplus, \otimes, 1\}$-systems is undecidable, and second, we will reduce this problem to both the existential membership problems for recurrent $\{\cup, \cap, \oplus, \otimes\}$-systems and recurrent $\{^-, \oplus, \otimes\}$-systems. The first step is based on the fact that it is undecidable whether a given Diophantine equation has a natural solution.

**Definition 18** *A **Diophantine equation** is an equation on variables $x_1, \ldots, x_k$ of the form $p(x_1, \ldots, x_k) = q(x_1, \ldots, x_k)$ where $p$ and $q$ are polynomials in $x_1, \ldots, x_k$ with natural coefficients. The problem* DIOPHANTINE *asks whether a Diophantine equation has a solution in natural numbers.*

**Theorem 7.16 (Matiyasevich, [59])**
DIOPHANTINE *is undecidable.*

The existential membership problem for recurrent $\{\text{gr}, \oplus, \otimes, 1\}$-systems is defined in the usual style, and it is denoted as $\text{M}_{ex}(\text{gr}, \oplus, \otimes, 1)$.

**Lemma 7.17** DIOPHANTINE $\leq_m^{\text{L}} \text{M}_{ex}(\text{gr}, \oplus, \otimes, 1)$.

**Proof:** Let $p(x_1, \ldots, x_k) = q(x_1, \ldots, x_k)$ be an instance of DIOPHANTINE. It holds that $(x_i)^{r+a} = (x_i)^r \cdot (x_i)^a$ for all $r$ and $a$. So, in logarithmic space, we can compute a circuit representation for $p$ and $q$ and a $\{\text{gr}, \oplus, \otimes, 1\}$-function $f_{p=q}$ such that

$$f_{p=q}(\{a_1\}, \ldots, \{a_k\}) = 1 \iff p(a_1, \ldots, a_k) = q(a_1, \ldots, a_k).$$

Now, we obtain the recurrent $\{\text{gr}, \oplus, \otimes, 1\}$-system $S$ from $S_E^k$ by adding a new component $(f_{p=q}, 0)$, and $f_{p=q}$ depends on the variables of $S_E^k$ associated with the functions $f_m^{(1)}, \ldots, f_m^{(k)}$. It holds that $1 \in [S]$ if and only if there are $a_1, \ldots, a_k \in \mathbb{N}$ such that $p(a_1, \ldots, a_k) = q(a_1, \ldots, a_k)$.

■

**Lemma 7.18** *(1)* $\text{M}_{ex}(\text{gr}, \oplus, \otimes, 1) \leq_m^{\text{L}} \text{M}_{ex}(\cup, \cap, \oplus, \otimes)$.
*(2)* $\text{M}_{ex}(\text{gr}, \oplus, \otimes, 1) \leq_m^{\text{L}} \text{M}_{ex}(^-, \oplus, \otimes)$.

**Proof:** The major part of the proof is dedicated to representing function gr as a $\{^-, \oplus, \otimes\}$-function and a $\{\cup, \cap, \oplus, \otimes\}$-function. However, this is too restrictive. In the former case, we will use further constant functions, in the latter case, we give a representation that strongly depends on a given recurrent system.

We begin by expressing gr as a $\{^-, \oplus, \otimes, 0, 1\}$-function. Consider the following definitions:

$$\begin{aligned}
f_1(x,z) &=_{\text{def}} \overline{\overline{\overline{\overline{0} \oplus z} \oplus ((x \oplus 1) \otimes (\overline{1} \oplus \overline{1 \oplus 1}))} \otimes 0} \\
f_2(x,z) &=_{\text{def}} \overline{(1 \oplus 1) \otimes f_1(x,z)} \otimes (((1 \oplus 1) \otimes f_1(x,z)) \oplus 1) \\
f_3(x,z) &=_{\text{def}} \overline{\overline{\overline{f_1(x,z)} \oplus 1} \oplus f_2(x,z)}\,.
\end{aligned}$$

Let $A, B \subseteq \mathbb{N}$ be non-empty set. Let $a_0 =_{\text{def}} \min A$ and $b_0 =_{\text{def}} \min B$. Observe that

$$\{0, \ldots, b_0\} = \overline{\{1, 2, \ldots\} \oplus \{b_0\}} = \overline{\overline{0} \oplus B}\,.$$

Furthermore, $\mathbb{N} = \overline{1} \oplus \overline{1 \oplus 1}$, and

$$\{a_0{+}1\} \otimes \mathbb{N} = \{0, a_0{+}1, 2a_0{+}2, \ldots\} \subseteq (A \oplus 1) \otimes (\overline{1} \oplus \overline{1 \oplus 1}) \subseteq \mathbb{N} \setminus \{a_0\}\,.$$

Thus,

$$f_1(A,B) = \begin{cases} \mathbb{N} & \text{, if } a_0 \le b_0 \\ \mathbb{N} \oplus 1 & \text{, if } a_0 > b_0\,. \end{cases}$$

Distinguishing the two cases, we obtain

$$f_2(A,B) = \begin{cases} \{1,3,5,7,9,\ldots\} & \text{, if } a_0 \le b_0 \\ \{0,3,5,7,9,\ldots\} & \text{, if } a_0 > b_0\,, \end{cases}$$

so that we obtain

$$\begin{aligned}
f_3(A,B) &= \begin{cases} \overline{\mathbb{N} \oplus \{1,3,5,7,9,\ldots\}} & \text{, if } a_0 \le b_0 \\ \overline{(\mathbb{N} \setminus \{1\}) \oplus \{0,3,5,7,9,\ldots\}} & \text{, if } a_0 > b_0 \end{cases} \\
&= \begin{cases} \{0\} & \text{, if } a_0 \le b_0 \\ \{1\} & \text{, if } a_0 > b_0\,, \end{cases}
\end{aligned}$$

which shows that $f_3(A,B) = \text{gr}(A,B)$ for all non-empty $A, B \subseteq \mathbb{N}$. So, let $S$ be a recurrent $\{\text{gr}, \oplus, \otimes, 1\}$-system. By replacing every occurrence of gr in the functions of $S$ by function $f_3$, we obtain a recurrent $\{^-, \oplus, \otimes, 0, 1\}$-system. Adding two further components representing the constant functions 0 and 1, we obtain a recurrent $\{^-, \oplus, \otimes\}$-system $S'$ for which holds $S(t) = S'(t)$ for every $t \ge 0$.

Now, we show that gr can be expressed as a $\{\cup, \cap, \oplus, \otimes, 0, 1\}$-function in a restricted sense, that is explained later. We have to make some preliminary considerations. Let $f$ be an $n$-ary $\{\text{gr}, \oplus, \otimes, 1\}$-function represented by a circuit containing $\nu$ vertices. Observe that for finite and non-empty sets $B_1, \ldots, B_n \subseteq \mathbb{N}$ it holds that

$$f(B_1, \ldots, B_n) \subseteq \{0, \ldots, (b_0{+}1)^{2^\nu}\}$$

where $b_0 =_{\text{def}} \max(B_1 \cup \cdots \cup B_n)$. Let

$$\text{el}(x) =_{\text{def}} (((1 \oplus x) \otimes x) \oplus x)\,.$$

Then, for every $a \in \mathbb{N}$ holds:

$$\text{el}(\{0,\ldots,a\}) = \{0,\ldots,a(a{+}2)\}\,.$$

Hence, we can generate function $\text{el}_\nu(x)$ where

$$\begin{aligned}\text{el}_0(x) &=_{\text{def}} x\\ \text{el}_{r+1}(x) &=_{\text{def}} \text{el}(\text{el}_r(x))\end{aligned}$$

in space linear in $\nu$, and for every $a \in \mathbb{N}$ holds $\{0,\ldots,a^{2^\nu}\} \subseteq \text{el}_\nu(\{0,\ldots,a\})$.

Let $S = (\mathcal{F}, A)$ be a recurrent $\{\text{gr}, \oplus, \otimes, 1\}$-system of dimension $n$. Let $a_0 =_{\text{def}} \max A$. In space linear in the length of $a_0$ we can generate a $\{\oplus, 0, 1\}$-function $g(\mathbf{x})$, where $\mathbf{x} = (x_1,\ldots,x_n)$, in the following way such that $\{0,\ldots,a_0\} \subseteq g(A)$:

$$\begin{aligned}\{0,1\} \oplus \{0,1\} &= \{0,1,2\}\\ \{0,1,2\} \oplus \{0,1,2\} &= \{0,1,2,3,4\}\\ \{0,1,2,3,4\} \oplus \{0,1,2,3,4\} &= \{0,1,\ldots,6,7,8\}\,,\end{aligned}$$

and so on. Let $\nu$ denote the largest number of vertices of the circuit representations of the functions of $S$. Let $S'$ emerge from $S$ by adding the components

$$(\langle f_{n+1}(\mathbf{x}') =_{\text{def}} x_{n+1}, f_{n+2}(\mathbf{x}') =_{\text{def}} x_{n+2}, f_{n+3}(\mathbf{x}') =_{\text{def}} \text{el}_\nu(g(x_{n+1} \cup x_{n+3}))\rangle, (1,0,0))$$

where $\mathbf{x}' =_{\text{def}} (x_1,\ldots,x_n,x_{n+1},x_{n+2},x_{x_3})$. Then, for every $t \geq 0$, $S'[f_{n+3}](t+1)$ contains all numbers that are not larger than any number that can be generated by a $\{\text{gr}, \oplus, \otimes, 1\}$-function represented by a circuit on at most $\nu$ vertices applied to $\mathcal{F}(t)$. The combination of all these results yields the following identity. For every $a, b \in \mathbb{N}$ that can appear in $\mathcal{F}(t)$:

$$\begin{aligned}\text{gr}(\{a\},\{b\}) =& \Big((\{b\} \cap (\{a\} \oplus f_{n+2}(t{+}1))) \otimes 0\Big)\\ &\cup \Big(((\{a\} \cap (\{b\} \oplus f_{n+2}(t{+}1) \oplus 1)) \otimes 0) \oplus 1\Big)\,.\end{aligned}$$

Obtain $S''$ from $S'$ by replacing every occurrence of gr by the above defined term, then by replacing every occurrence of 0 and 1 by $x_{n+2}$ and $x_{n+1}$, respectively, and finally by making $f_n$ the output function of $S''$. Hence, for every $t \geq 0$, $S(t) = S''(t)$, and $S''$ can be obtained using only logarithmic space.

■

**Theorem 7.19** *$\text{M}_{ex}(\cup, \cap, \oplus, \otimes)$ and $\text{M}_{ex}(^{-}, \oplus, \otimes)$ are undecidable.*

**Proof:** Undecidability is an immediate consequence of Lemmata 7.18 and 7.17 and Theorem 7.16.

■

# Part III

# Minimal triangulations

*Every graph defines the set of its minimal triangulations. A minimal triangulation of a graph $G$ is a chordal spanning supergraph of $G$ such that it does not have a proper subgraph that is also a chordal spanning supergraph of $G$. Since there is only a finite number of graphs of fixed vertex number, the set of minimal triangulations of a graph is finite. Nevertheless, the problem, given a pair $(G, H)$ of graphs, to decide whether $H$ is a minimal triangulation of $G$, is not trivial. This problem will be called the min-Tri membership problem for arbitrary graphs. Starting only from the definition, it is not clear whether the min-Tri membership problem is polynomial-time solvable. However, there is a characterisation of minimal triangulations that immediately implies a polynomial-time decision algorithm.*

*Instead of investigating the general min-Tri membership problem, we will concentrate on restricted versions. The restrictions will always affect the "first" input graph, i.e., the graph that defines the set of minimal triangulations. We will consider the min-Tri membership problem for $2K_2$-free graphs, i.e., for graphs that do not contain the $2K_2$ as induced subgraph, permutation graphs and AT-free claw-free graphs. In all three cases, we will obtain linear-time solutions, if we assume that the first input graph belongs to the requested graph class. Otherwise, i.e., if it must be checked* *whether the first input graph has the required properties, recognition algorithms have to be applied, and the min-Tri membership problems may become more difficult.*

*The linear-time algorithm for solving a restricted version of the min-Tri membership problem is based on efficient characterisations of minimal triangulations of graphs that belong to the considered graph class. These characterisations are of different types. In case of $2K_2$-free graphs, the vertex set of a minimal triangulation of such a graph can be partitioned into two sets, and one of the partition classes must fulfill an easily checkable property. For permutation graphs, we will define a special acyclic graph such that every maximal path of this acyclic graph represents a minimal triangulation in a way that allows efficient construction of the represented triangulation. Finally, in case of AT-free claw-free graphs, minimal triangulations are characterised by vertex orderings that can be generated by a special algorithm applied to the given graph. So, every graph class requires a different approach for solving the restricted versions of the min-Tri membership problem.*

*Besides solutions for our central problems, we will obtain a number of results that will appear only as auxiliary tools but are interesting in their own right. We will present an easy recognition algorithm for proper interval graphs. Proper interval graphs are special chordal graphs. The algorithm is multi-sweep breadth first search based and generates a vertex ordering with a property that characterises proper interval graphs, if the input graph is actually a proper interval graph; otherwise, i.e., if the input graph is not a proper interval graph, the algorithm outputs a vertex set which proves that the input graph cannot be a proper interval graph. Such algorithms are called "certifying", since they do not only produce an answer but prove their decisions by appropriate certificates.*

*In case of permutation graphs, we derive simple and linear-time algorithms for computing the treewidth and minimum fill-in of permutation graphs. These algorithms are modified shortest paths algorithms and work on the acyclic graph whose maximal paths represent the set of minimal triangulations of a permutation graph. For $2K_2$-free graphs, treewidth and minimum fill-in do not become linear-time computable, but we will at least obtain polynomial-time algorithms.*

# Chapter 8
# Chordal graphs and minimal triangulations

Minimal triangulations are chordal graphs. A graph is called chordal if it does not contain a chordless cycle of length at least 4. A cycle is chordless if only consecutive vertices are adjacent. Chordal graphs are very interesting graphs to study, and they provide many characterisations using basic graph notions. Among those, three characterisations are of special interest. So, vertices of chordal graphs can be ordered such that the resulting ordering has an easily verifiable property (these orderings are called perfect elimination schemes). Chordal graphs are exactly those graphs whose minimal separators are cliques. This establishes a beautiful connection to the wide field of separators. Finally, chordal graphs are the intersection graphs of families of subtrees of trees. Hence, chordal graphs appear also in the context of graph classes with nice geometric models.

In this chapter, we will present general definitions and results that are needed for this part of the thesis. We will first define the notion of minimal separators and give useful characterisations and properties. After that, we will consider the class of chordal graphs. Especially, we will focus on characterisations of chordal graphs that will appear later in modified form for characterisations of subclasses of chordal graphs. Then, we will be prepared for defining minimal triangulations. A minimal triangulation of a graph is obtained by adding an inclusion-minimal set of edges to make the given graph chordal. We will give two well-known and useful characterisations of minimal triangulations. After that, we will define the min-Tri membership problem for arbitrary graphs, which is the general version of the problems that we wish to solve in the subsequent chapters for special graph classes. The last section of this chapter will be dedicated to three graph parameters that turn out to be related to triangulations of graphs. Further definitions and results are stated in the subsequent chapters, where they are needed.

## 8.1 Minimal vertex separators

For analysing the structure of graphs, it is useful to partition the graph into smaller pieces. Some concepts are known, and the most popular is based on separators. And for our purposes, special separators are of high importance. Let $G = (V, E)$ be a graph, and let $S \subseteq V$. By $G \setminus S$, we denote the subgraph of $G$ induced by $V \setminus S$. If $S = \{x\}$ for some vertex $x$ of $G$, we also write $G{-}x$ instead of $G \setminus \{x\}$.

**Definition 19** *Let $G = (V, E)$ be a graph, and let $a, b \in V$ be vertices of $G$. A vertex set $S \subseteq V$ of $G$ is an $a, b$-**separator** of $G$ if and only if $a$ and $b$ are contained in different connected components of $G \setminus S$.*

Note that there may be vertices $a$ and $b$ of a graph $G = (V, E)$ for which there is no $a, b$-separator of $G$. This is the case if and only if $a$ and $b$ are adjacent in $G$. For non-adjacent $a$ and $b$, $V \setminus \{a, b\}$ is always an $a, b$-separator of $G$.

**Definition 20** *Let $G = (V, E)$ be a graph, and let $a, b \in V$ be vertices of $G$. A vertex set $S \subseteq V$ of $G$ is a **minimal** $a, b$-separator of $G$ if and only if no proper subset $S' \subset S$ of $S$ is an $a, b$-separator of $G$.*

Since a different notion of minimal $a, b$-separators may be defined via a cardinality property, our minimal $a, b$-separators are also called *inclusion-minimal.*

**Definition 21** *Let $G = (V, E)$ be a graph. A vertex set $S \subseteq V$ of $G$ is a **minimal separator** of $G$ if and only if there are vertices $a, b \in V$ of $G$ such that $S$ is a minimal $a, b$-separator of $G$.*

Minimal separators have a useful characterisation. Let $G = (V, E)$ be a graph, and let $S \subseteq V$ be a set of vertices of $G$. An *$S$-full component* of $G$ is the subgraph of $G$ induced by a vertex set $C \subseteq V$ where $C$ induces a connected component of $G \setminus S$ and every vertex in $S$ has a neighbour in $C$. The following lemma is easy and appears several times in the literature.

**Lemma 8.1 (folklore)**
*Let $G = (V, E)$ be a graph, and let $a$ and $b$ be vertices of $G$. A vertex set $S \subseteq V$ of $G$ is a minimal $a, b$-separator of $G$ if and only if $G$ has two (different) $S$-full components where the one contains $a$ and the other contains $b$.*

**Proof:** Let $S$ be a minimal $a, b$-separator of $G$. By definition, $a$ and $b$ appear in different connected components $C_a$ and $C_b$ of $G \setminus S$, respectively. Suppose there is $x \in S$ that has no neighbour in $C_a$. Every $a, b$-path in $G$ that contains $x$ contains another vertex from $S$. Hence, $S \setminus \{x\}$ is also an $a, b$-separator of $G$, which contradicts $S$ being minimal. A similar argumentation holds if $S$ contains a vertex that has no neighbour in $C_b$. For the converse, first note that $S$ is an $a, b$-separator of $G$. Furthermore, observe that, for every $x \in S$, there is an $a, b$-path in $G$ containing only $x$ as vertex from $S$. So, no proper subset of $S$ can separate $a$ and $b$, and $S$ is minimal.
■

**Corollary 8.2** *Let $G = (V, E)$ be a graph, and let $S \subseteq V$ be a vertex set of $G$. Then, $S$ is a minimal separator of $G$ if and only if $G$ has two $S$-full components.*

**Proof:** Let $S$ be a minimal separator of $G$. By definition, there are vertices $a$ and $b$ of $G$ such that $S$ is a minimal $a, b$-separator of $G$. Due to Lemma 8.1, $a$ and $b$ are contained in two $S$-full components of $G$. For the converse, let $C_1$ and $C_2$ be two $S$-full components of $G$. Then, for every $a \in V(C_1)$ and $b \in V(C_2)$, $S$ is a minimal $a, b$-separator of $G$ due to Lemma 8.1, hence a minimal separator of $G$.
■

An interesting and useful relation over the set of minimal separators of a graph is the crossing relation.

**Definition 22** *Let $G = (V, E)$ be a graph, and let $S_1, S_2 \subseteq V$ be minimal separators of $G$. We say that $S_1$ and $S_2$* ***cross*** *if and only if $S_2$ contains vertices of two connected components of $G \setminus S_1$.*

It is clear that crossing can also be defined for arbitrary separators; we only use the case minimal separators.

**Lemma 8.3 (Parra and Scheffler, [69])**
*Let $G = (V, E)$ be a graph, and let $S_1, S_2 \subseteq V$ be minimal separators of $G$. Then, $S_1$ and $S_2$ cross if and only if $S_2$ and $S_1$ cross, i.e., the crossing relation is symmetric.*

**Proof:** By symmetry, it suffices to prove only one implication. Let $S_1$ and $S_2$ cross, which means that there are vertices $a$ and $b$ in $S_2$ and (different) connected components $C_a$ and $C_b$ of $G \setminus S_1$ such that $C_a$ contains $a$ and $C_b$ contains $b$. Obviously, $S_1$ is an $a, b$-separator of $G$. Since $S_2$ is a minimal separator of $G$, there are two $S_2$-full components $D_1$ and $D_2$ of $G$, and $a$ and $b$ have neighbours in $D_1$ and in $D_2$. Then, $S_1$ must contain vertices from $D_1$ and $D_2$, since otherwise $G \setminus S_1$ would contain an $a, b$-path and $S_1$ would not separate $a$ and $b$.

■

The more interesting case for minimal separators is, if they do not cross. In this case, we speak of *non-crossing* minimal separators.

## 8.2 Getting to know chordal graphs

The class of chordal graphs is a very interesting graph class. In the history of graph theory, the class of chordal graphs plays an important role, since it has been among the first to be known a class of perfect graphs (perfect graphs are only mentioned here, that is why we do not define the term and refer to the book by Golumbic, [35], for more information and further historic remarks). Also, a lot of basic problems, such as computing the clique and independent set numbers, are easy for chordal graphs. The structure of chordal graphs is closely related to trees in a certain sense, as we will see at the end of this section, and this property is useful in the design of algorithms. In this section, we will summarise some important structural results for chordal graphs.

**Definition 23** *A graph $G = (V, E)$ is* ***chordal*** *if and only if $G$ does not contain a chordless cycle of length greater than 3.*

Minimal separators and chordal graphs become related by two somewhat surprising characterisations.

**Theorem 8.4 (Dirac, [25])**
*A graph is chordal if and only if all its minimal separators are cliques.*

**Proof:** Let $G = (V, E)$ be a graph. Let $S \subseteq V$ be a minimal separator of $G$. If $S$ is not a clique in $G$, there are two non-adjacent vertices $u$ and $v$ in $S$. Due to Lemma 8.1, $u$ and $v$ have neighbours in two connected components of $G \setminus S$, and there are two $u, v$-paths $P_1$ and $P_2$ in $G$ such that they form a chordless cycle of length at least 4 in $G$. Hence, $G$ is not chordal. Conversely, if $G$ is not chordal, there is a chordless cycle $C$ of length at least 4 in $G$. Let $x$ and $z$ be vertices in $C$ that are non-adjacent. Let $u$ and $v$ be the neighbours of $x$ in $C$. Then, $S' =_{\text{def}} (V \setminus V(C)) \cup \{u, v\}$ is an $x, z$-separator of $G$. There is $S \subseteq S'$ that is a minimal $x, z$-separator of $G$. Note that $S$ must contain $u$ and $v$. Hence, $S$ is not a clique in $G$.

■

**Corollary 8.5 (Parra and Scheffler, [69])**
*A graph $G = (V, E)$ is chordal if and only if $G$ does not have minimal separators that cross.*

**Proof:** Let $G$ have two minimal separators $S_1$ and $S_2$ such that $S_1$ and $S_2$ cross, i.e., there are vertices $u$ and $v$ in $S_2$ that appear in two different connected components of $G \setminus S_1$. Then, $u$ and $v$ are not adjacent in $G$, and $G$ is not chordal by Theorem 8.4. For the converse, let $G$ be not chordal, and let $C = (x_1, \ldots, x_r)$ be a chordless cycle of length $r \geq 4$ in $G$. Since $x_1$ and $x_3$ are non-adjacent in $G$, there is a minimal $x_1, x_3$-separator $S$ of $G$. It holds that $S$ contains $x_2$ and another vertex $x'$ from $\{x_4, \ldots, x_r\}$. Since $x_2$ and $x'$ are non-adjacent in $G$, $V \setminus \{x_2, x_4, \ldots, x_r\}$ is an $x_2, x'$-separator of $G$, that contains only $x_1$ and $x_3$ of cycle $C$. Hence, there is a minimal $x_2, x'$-separator $S'$ of $G$ containing $x_1$ and $x_3$. Then, $S'$ contains two vertices from two different connected components of $G \setminus S$, i.e., $S$ and $S'$ cross.

■

Interestingly, chordal graphs can also be characterised by special vertex orderings. Vertex orderings play an important role in this part of the thesis, since they often imply easy and efficient algorithms.

**Definition 24** *Let $G = (V, E)$ be a graph, and let $u$ be a vertex of $G$. We say that $u$ is **simplicial** in $G$ if and only if the neighbourhood of $u$ is a clique in $G$.*

We give a first connection between simplicial vertices and chordality. In fact, simplicial vertices do not influence the chordality of a graph.

**Lemma 8.6** *Let $G = (V, E)$ be a graph, and let $u$ be a simplicial vertex of $G$. Then, $G$ is chordal if and only if $G-u$ is chordal.*

**Proof:** Let $G$ be chordal. Then, for every vertex $x$ of $G$, $G-x$ is chordal. Conversely, let $G$ be not chordal, and let $C = (x_1, \ldots, x_r)$ be a chordless cycle of length $r \geq 4$ in

$G$. Observe that every vertex in $C$ has two non-adjacent neighbours in $G$. Hence, no simplicial vertex of $G$ is contained in $C$, and $C$ is a chordless cycle in $G{-}u$.

■

Another interesting connection between chordal graphs and simplicial vertices is shown by the following lemma. It says that making simplicial an arbitrary vertex of a chordal graph preserves chordality.

**Lemma 8.7 (Rose, Tarjan and Lueker, [76])** *Let $G = (V, E)$ be a chordal graph, and let $u$ be a vertex of $G$. Let*

$$F =_{\mathrm{def}} \{vw : v, w \in N_G(u) \text{ and } v \neq w \text{ and } vw \notin E\}\,.$$

*Then, $G' =_{\mathrm{def}} G \cup F$ is chordal.*

**Proof:** Observe that $u$ is simplicial in $G'$. Suppose $G'$ contains a chordless cycle $C$ of length at least 4. Since $u$ is simplicial in $G'$, $C$ cannot contain $u$. If $C$ contains at most one neighbour of $u$, $C$ is a chordless cycle in $G$, which is not possible by chordality of $G$. Hence, $C$ contains exactly two neighbours $v$ and $w$ of $u$, and $v$ and $w$ are adjacent in $G'$. But then, cycle $C'$ that is a modification of $C$ where $u$ is set between $v$ and $w$ is a chordless cycle in $G$ of length at least 5, which contradicts $G$ being chordal. Thus, $G'$ is chordal.

■

Set $F$ of Lemma 8.7 is called the *deficiency* of vertex $u$ in $G$ by Rose, Tarjan, Lueker [76].

So far, we know that chordal graphs and simplicial vertices are related in some sense. It remains, however, open which chordal graphs contain simplicial vertices. In brief, it turns out that every chordal graph contains simplicial vertices.

**Theorem 8.8 (Dirac, [25])**
*Let $G = (V, E)$ be a chordal graph that is not complete. Then, $G$ has two non-adjacent simplicial vertices.*

**Proof:** We prove the statement by induction over the number of vertices of the graphs. It is clear that the statement holds for all graphs on at most three vertices, since all these graphs are chordal, and each non-complete graph on at most three vertices contains two non-adjacent vertices whose neighbourhoods are cliques. So, let $G$ have at least four vertices. Let $u$ be a vertex of $G$ such that $u$ is not simplicial. Such a vertex exists, since $G$ is assumed to be non-complete. Let $v$ and $w$ be non-adjacent neighbours of $u$. Then, there is a minimal $v, w$-separator $S$ of $G$, and $S$ is a clique due to Theorem 8.4. Let $C_1$ and $C_2$ induce two different $S$-full components of $G$ where we assume, without loss of generality, that $C_1$ has at most as many vertices as $C_2$. We distinguish four cases:

(A) $C_1$ as well as $C_2$ contain at least two vertices. Let $G_1' =_{\mathrm{def}} G[C_1 \cup S]$ and $G_2' =_{\mathrm{def}} G[C_2 \cup S]$. Let $x_1$ and $x_2$ be new vertices. Let $G_1$ emerge from $G_1'$ by adding $x_1$ and making it adjacent with every vertex from $S$; similarly, obtain $G_2$ from $G'_2$ by adding $x_2$. Note that $G_1$ and $G_2$ contain less vertices than $G$. Furthermore, no vertex from $S$ is simplicial in $G$, $G_1$ or $G_2$. Hence, by induction hypothesis, $G_1$ and $G_2$ each contain two non-adjacent simplicial vertices, and two of them are simplicial in $G$.

(B) $C_1$ as well as $C_2$ contain exactly one vertex. Then, these two vertices are simplicial.

(C) $C_1$ contains exactly one vertex and $C_2$ contains at least two vertices where $C_2$ is a clique in $G$. Let $x$ and $z$ be vertices from $C_2$. If $N_G[x] \not\subseteq N_G[z]$ and $N_G[z] \not\subseteq N_G[x]$, $G$ contains a cycle of length 4, which is not possible. Hence, $C_2$ contains a vertex whose closed neighbourhood is contained in the closed neighbourhood of every other vertex from $C_2$, and so this vertex is simplicial.

(D) $C_1$ contains exactly one vertex and $C_2$ contains at least two vertices where $C_2$ is not a clique in $G$. So, there are two non-adjacent vertices $x$ and $z$ in $C_2$. Let $T$ be a minimal $x, z$-separator of $G$, and let $D_1$ and $D_2$ be $T$-full components of $G$. If one of these components contains exactly one vertex, this vertex is different from the vertex in $C_1$ and also non-adjacent with it, so that we have found two non-adjacent simplicial vertices in $G$. Otherwise, $D_1$ and $D_2$ each contain at least two vertices, and we can apply the construction of case (A) using $D_1$ and $D_2$ instead of $C_1$ and $C_2$, respectively.

Every of the four cases shows the existence of two non-adjacent simplicial vertices in $G$. The construction of case (D) relies on the fact that minimal separators of chordal graphs are non-crossing due to Corollary 8.5.

■

The existence of simplicial vertices in chordal graphs is the foundation of a characterisation of chordal graphs by vertex orderings.

**Definition 25** *Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. We say that $\sigma$ is a* ***perfect elimination scheme*** *for $G$ if and only if for every triple $u, v, w$ of vertices of $G$ holds:*

$$u \prec_\sigma v \prec_\sigma w \text{ and } uv \in E \text{ and } uw \in E \quad \Longrightarrow \quad vw \in E\,.$$

It follows from the definition of perfect elimination schemes that, for every vertex, the set of neighbours to the right forms a clique in the graph. Hence, for $G = (V, E)$ a graph on $n$ vertices and $\sigma = \langle x_1, \ldots, x_n \rangle$ a vertex ordering for $G$, $\sigma$ is a perfect elimination scheme for $G$ if and only if $x_i$ is simplicial in $G[\{x_i, \ldots, x_n\}]$ for every $i \in \{1, \ldots, n\}$. It will be one of our main goals to generate vertex orderings for special graphs that are as close to perfect elimination schemes as possible.

**Corollary 8.9 (Rose, [75])**
*A graph is chordal if and only if it has a perfect elimination scheme.*

**Proof:** Let $G = (V, E)$ be a graph. Let $\sigma$ be a perfect elimination scheme for $G$. Suppose there is a chordless cycle $C = (x_1, \ldots, x_r)$ in $G$ of length $r \geq 4$. Without loss of generality, $x_1$ is leftmost with respect to $\sigma$, i.e., $x_1 \prec_\sigma x_i$ for all $i \in \{2, \ldots, r\}$. Note that $x_2$ and $x_r$ are neighbours of $G$, that are to the right of $x_1$; we may assume $x_1 \prec_\sigma x_2 \prec_\sigma x_r$. Due to the definition of perfect elimination schemes, $x_2$ and $x_r$ must be adjacent in $G$, which contradicts the assumption. Hence, $G$ is chordal.

For the converse, let $G$ be chordal. We give an inductive proof. If $G$ has exactly one vertex $u$, then $\langle u \rangle$ is a perfect elimination scheme for $G$. For the induction step, let $G$ have at least two vertices. Due to Theorem 8.8, $G$ has a simplicial vertex $u$. Consider $G' =_{\text{def}} G - u$. Certainly, $G'$ is chordal and has less vertices than $G$. There is a perfect elimination scheme $\langle x_1, \ldots, x_r \rangle$ for $G'$ by induction hypothesis. Then, $\langle u, x_1, \ldots, x_r \rangle$ is a perfect elimination scheme for $G$.

■

It is clear that a simplicial vertex of a graph can be found in polynomial time, if there is one. However, it is not clear at first glance, whether this is possible also in linear time. An even stronger result holds.

**Theorem 8.10 (Rose, Tarjan and Lueker, [76])**
*There is a linear-time algorithm that, on input graph $G$, generates a perfect elimination scheme for $G$ if and only if $G$ is chordal.*

Another linear-time algorithm for computing perfect elimination schemes for chordal graphs has been presented by Tarjan and Yannakakis [81].

**Corollary 8.11 (Rose, Tarjan and Lueker, [76])**
*There is a linear-time algorithm that recognizes chordal graphs.*

**Proof:** Let $G = (V, E)$ be a graph. Due to Theorem 8.10, there is a linear-time algorithm for generating a vertex ordering $\sigma$ that is a perfect elimination scheme for $G$ if and only if $G$ is chordal. So, it remains to verify in linear time whether $\sigma$ is a perfect elimination scheme for $G$. It can be shown by induction that $\sigma$ is a perfect elimination scheme if and only if $\sigma$ is a *reduced elimination scheme* for $G$, i.e., for all $u, v \in V$,

$$\text{lmr}(u) \prec_\sigma v \text{ and } uv \in E \quad \Longrightarrow \quad \text{lmr}(u)v \in E$$

where $\text{lmr}(u)$ denotes the *leftmost right neighbour* of $u$ with respect to $\sigma$, if there is a neighbour $x$ of $u$ such that $u \prec_\sigma x$; otherwise, $\text{lmr}(u) =_{\text{def}} u$. Since $G$ is given by adjacency lists, and these lists can be ordered according to $\sigma$ in linear time, it can be tested in linear time whether $\sigma$ is a reduced elimination scheme. We obtain a linear-time recognition algorithm for chordal graphs.

■

A characterisation of chordal graphs of a completely different flavour, at first glance, is given by the following theorem. It shows that chordal graphs have a nice geometric representation and, moreover, that they have a tree-like structure.

**Theorem 8.12 (Buneman, [16]; Gavril, [32]; Walter, [85])**
*Let $G = (V, E)$ be a graph. Then, $G$ is chordal if and only if there is a tree $T$ and a family $\mathcal{T}$ of subtrees of $T$ such that $G$ is the intersection graph of $\mathcal{T}$.*

## 8.3 Characterisations of minimal triangulations of arbitrary graphs

The main objects that we wish to study and to deal with are minimal triangulations of graphs. They appear in different contexts, and in the following sections, we will discuss two of them. In this section, we will give definitions, two important characterisations of minimal triangulations of arbitrary graphs and further properties.

**Definition 26** *Let $G = (V, E)$ be a graph, and let $H = (W, F)$ be a graph. We call $H$ a* ***triangulation*** *of $G$ if and only if $H$ is a spanning supergraph of $G$ and chordal.*

In other words, a triangulation of a graph $G$ is a graph defined on the vertex set of $G$ and that is obtained from $G$ by adding further edges to make it chordal. Since the complete graph on the vertex set of $G$ is a spanning chordal supergraph of $G$, a triangulation of $G$ exists. If $G$ is not chordal, every triangulation of $G$ is a proper supergraph of $G$; if $G$ is chordal it is also a triangulation of itself. A small example of a graph and one of its triangulations is depicted in Figure 16. It may not be obvious at first glance that the right side graph is chordal, but using Corollary 8.9, chordality can be proven easily.

**Definition 27** *Let $G = (V, E)$ be a graph, and let $H = (V, F)$ be a triangulation of $G$. We call $H$ a* ***minimal*** *triangulation of $G$ if and only if there is no proper subgraph $H'$ of $H$ that is a triangulation of $G$.*

We observe that minimality in this context is defined as a set-inclusion property. That is why in the literature, the above defined minimal triangulations may also be called *inclusion-minimal* triangulations. The definition of minimal triangulations dates back to the fundamental work of Rose, Tarjan, Lueker about perfect elimination processes [76]. From an algorithmic point of view, it is interesting as well as important to note that minimal triangulations of a graph with only a few edges may have many edges. Famous such examples are complete bipartite graphs. A graph is *bipartite* if its vertex set can be partitioned into two independent sets, and a bipartite graph is *complete* if it contains all edges with endpoints in the two different independent sets. It is not hard to see that minimal triangulations of complete bipartite graphs are obtained by adding all edges between vertices of exactly one partition set. If one independent set contains only one vertex, the graph is already chordal.

For our first characterisation of minimal triangulations of arbitrary graphs, we state an important partial result.

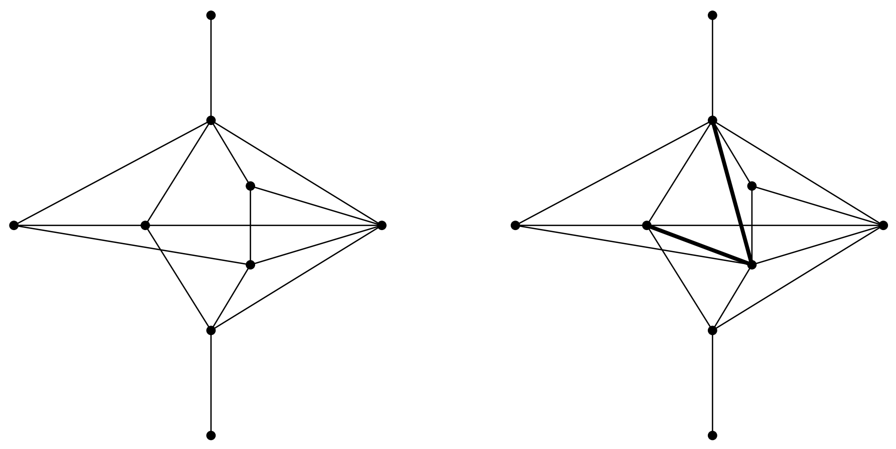


**Figure 16** The picture shows two graphs. The left side graph contains chordless cycles of length 4 and 5, and so it is not chordal. The right side graph looks like a copy of the left side graph, but it contains two additional edges, that are accentuated as thick lines. Because of these two additional edges, the right side graph is chordal, hence the right side graph is a triangulation of the left side graph.

**Lemma 8.13 (Rose, Tarjan and Lueker, [76])**
*Let $G = (V, E)$ be a chordal graph. Let $F$ be a non-empty set of edges for $G$ where $E \cap F = \emptyset$, and let $G' =_{\text{def}} G \cup F$. If $G'$ is chordal, then there is an edge $e \in F$ such that $G'{-}e$ is chordal.*

**Proof:** We prove the statement by induction over the number of vertices of $G$. If $G$ has at most three vertices, every graph on the vertex set of $G$ is chordal, and the statement holds. So, let $G$ and $G'$ have at least four vertices. If $G'$ contains a simplicial vertex $u$ such that $u$ has a neighbour $v$ in $G'$ and $uv \in F$, then $G'{-}uv$ is chordal, since $u$ remains simplicial in $G'{-}uv$ (Corollary 8.9).

Now, assume that $G'$ does not contain a simplicial vertex with the property above. Then, for every simplicial vertex $x$ of $G'$, $N_{G'}(x) = N_G(x)$. We define, for every vertex $x$ of $G'$, $D(x) =_{\text{def}} \{vw \notin E : v, w \in N_{G'}(x) \text{ and } v \neq w\}$. We first show that there is a simplicial vertex $x$ in $G'$ such that $F \not\subseteq D(x)$. Let $u$ be any simplicial vertex of $G'$. If $F \not\subseteq D(u)$, let $u$ be chosen. Otherwise, $F \subseteq D(u)$. It immediately follows that $F = D(u)$. Hence, $G'$ emerged from $G$ by making $u$ simplicial. Since the case above does not hold, every simplicial vertex of $G$ is simplicial in $G'$ and has the same neighbourhood. Let $v$ be a simplicial vertex of $G$. Then, $F \not\subseteq D(v) = \emptyset$, and we choose $v$ as $u$. So, $u$ is a simplicial vertex of $G'$, and $F \not\subseteq D(u)$. Consider the two graphs $H =_{\text{def}} (G{-}u) \cup D(u)$ and $H' =_{\text{def}} G'{-}u$. Note that $G \cup D(u)$ is chordal due to Lemma 8.7. Then, $H$ is also chordal, and $H'$ is a chordal supergraph of $H$. Since $H'$ has less vertices than $G'$, we can apply the induction hypothesis and obtain an edge $e \in F \setminus D(u)$ such that $H'{-}e$ is chordal. Hence, $G'{-}e$ is chordal.

■

From Lemma 8.13, it follows that, for every chordal graph $G$ that is not blank, $G$ contains an edge $e$ such that $G{-}e$ is also chordal. This is true since the blank graph on the vertex set of $G$ is clearly chordal.

The first characterisation of minimal triangulations of arbitrary graphs that we give here relies on Lemma 8.13.

**Theorem 8.14 (Rose, Tarjan and Lueker, [76])**
*Let $G = (V, E)$ and $H = G \cup F$ be graphs where $E \cap F = \emptyset$. Let $H$ be chordal. Then, the following statements are equivalent:*

*(1) $H$ is a minimal triangulation of $G$*

*(2) for every $e \in F$, $e$ is unique chord in a cycle of length 4 in $H$*

*(3) for every $e \in F$, the graph $H{-}e$ is not chordal.*

**Proof:** We prove the theorem by showing three implications.

(1)⇒(2)
Let $F \neq \emptyset$, and let $uv \in F$. Since $H$ is a minimal triangulation of $G$, $H{-}uv$ is not chordal. Then, there must be a chordless cycle $C$ of length greater than 3 in $H{-}uv$ that contains $u$ and $v$. If the length of $C$ is greater than 4, only one edge does not

suffice to make $H-uv$ chordal; hence, $C$ has length 4. Let $C = (u, x, v, z)$. It holds that $x$ and $z$ are non-adjacent in $H$, so that $uv$ is unique chord in the cycle $C$ in $H$.

(2)⇒(3)
Let $F \neq \emptyset$, and let $e \in F$. Since $e$ is unique chord in a cycle $C$ of length 4, $C$ does not have a chord in $H-e$. Hence, $H-e$ contains a chordless cycle of length at least 4, i.e., is not chordal.

(3)⇒(1)
Suppose there is a triangulation $H'$ of $G$ that is a proper subgraph of $H$. By Lemma 8.13, $H$ contains an edge $e$ that is not contained in $H'$ such that $H-e$ is chordal. This, however, contradicts the assumption about $H$, and so, $H$ is a minimal triangulation of $G$.

■

Statement (3) of Theorem 8.14 implies an easy polynomial-time algorithm for testing, given two graphs $G$ and $H$, whether $H$ is a minimal triangulation of $G$: simply check, for every edge $e$ in $H$ that is not an edge in $G$, whether $H-e$ is chordal. Chordality testing can be done in linear time, and $H$ can have at most $n^2$ many edges where $n$ is the number of vertices of $G$. This gives an $\mathcal{O}(n^4)$-time algorithm. In connection with Lemma 8.13, also a polynomial-time algorithm for making a triangulation of $G$ minimal follows. It suffices to delete edges as long as the new graph is chordal. If such an edge cannot be chosen, the obtained graph is a minimal triangulation of $G$.

Our second characterisation of minimal triangulations is a structural result, that relates minimal separators and minimal triangulations. It characterises sets of edges that have to be added to obtain minimal triangulations. For a graph $G$ and a set $S$ of vertices of $G$, *completing $S$ into a clique* means the construction of a new graph $G'$ that is obtained from $G$ by adding all edges between vertices from $S$ that are not contained in $G$, so that $S$ is a clique in the new graph. For a family $\mathfrak{F}$ of sets of vertices of $G$, completing all sets in $\mathfrak{F}$ into cliques means that every set of $\mathfrak{F}$ is completed into a clique separately.

**Theorem 8.15 (Parra and Scheffler, [69])**
*Let $G = (V, E)$ be a graph.*

*(1) Let $\mathfrak{S}$ be a maximal set of pairwise non-crossing minimal separators of $G$. Graph $H$ is obtained from $G$ by completing into cliques all separators in $\mathfrak{S}$. Then, $H$ is a minimal triangulation of $G$.*

*(2) Let $H$ be a minimal triangulation of $G$ and let $\mathfrak{S}$ be the set of minimal separators of $H$. Then, $\mathfrak{S}$ is a maximal set of pairwise non-crossing minimal separators of $G$ and $H$ originates from $G$ by completing into cliques the separators in $\mathfrak{S}$.*

Another structural property of graphs and their minimal triangulations is stated in the following theorem.

**Theorem 8.16 (Kloks, [47])**
*Let $G = (V, E)$ be a graph, and let $H$ be a minimal triangulation of $G$.*

*(1) For every pair $a, b$ of vertices of $H$, if $S \subseteq V$ is a minimal $a, b$-separator of $H$, then $S$ is a minimal $a, b$-separator of $G$.*

*(2) For every minimal separator $S$ of $H$ and every connected component $C$ of $H \setminus S$, $C$ is a connected component of $G \setminus S$.*

Finally, we consider the neighbourhoods of vertices in a graph and its minimal triangulations. As a special case, we obtain that simplicial vertices of a graph are simplicial in every of its minimal triangulations. Remember that the existence of simplicial vertices is not restricted to only chordal graphs.

**Lemma 8.17** *Let $G = (V, E)$ be a graph, and let $H$ be a minimal triangulation of $G$. Let $u$ be a vertex of $G$. If $N_G(u)$ is a clique in $H$, then $N_G(u) = N_H(u)$.*

**Proof:** By definition of minimal triangulations, $N_G(u) \subseteq N_H(u)$ obviously holds. Let $N_G(u)$ be a clique in $H$, and let $F =_{\text{def}} \{uv : v \in N_H(u) \setminus N_G(u)\}$ be non-empty. Consider $H' =_{\text{def}} H \setminus F$. Certainly, $H'$ is a proper spanning subgraph of $H$ and a supergraph of $G$. Assume $C$ is a chordless cycle of length at least 4 in $H'$. Then, $C$ contains $u$, since otherwise, $C$ would be a chordless cycle in $H'-u = H-u$ and hence in $H$. But then, $C$ also contains two neighbours of $u$, and since $u$ is simplicial in $H'$, these two neighbours are adjacent. Therefore, $C$ cannot be chordless or is of length only 3. Hence, $H'$ is chordal, and $H$ cannot be a minimal triangulation of $G$.

■

## 8.4 The min-Tri membership problem for arbitrary graphs

Graphs may have several, in most cases many, minimal triangulations. So, the set of minimal triangulations of a graph may become of considerable size, and it may be "difficult" to decide, given two graphs $G$ and $H$, whether $H$ is a minimal triangulation of $G$. We highlighted the word *difficult* since we have already seen that the question is polynomial-time decidable using the characterisation of Theorem 8.14, and the polynomial is not too large. However, we look for faster algorithms. But first, let us formally define our problem. The *min-Tri membership problem for arbitrary graphs* is the set of pairs $(G, H)$ of graphs where $H$ is a minimal triangulation of $G$.

There are three algorithms known that solve the min-Tri membership problem. Though, two of them are originally designed for another, more general purpose, namely making a triangulation of a graph minimal. So, observe that it is easy to decide, given a pair $(G, H)$ of graphs, whether $H$ is a triangulation of $G$: by definition, $H$ is a triangulation of $G$ if and only if $H$ is a chordal spanning supergraph of $G$. Chordality and being spanning supergraph can be decided in linear time (linear time chordality testing is due to Theorem 8.10).

For testing, given a pair $(G, H)$ of graphs where $H$ is a triangulation of $G$, whether $H$ is a minimal triangulation of $G$, Blair, Heggernes, Telle propose an algorithm that iteratively generates a vertex ordering of $G$ that defines a minimal triangulation of $G$ that is a subgraph of $H$ [6]. Another algorithm is given by Dahlhaus, that is also based on generating vertex orderings [23]. The third algorithm simply applies the characterisation of minimal triangulations given in Theorem 8.14. Ibarra addressed the problem of maintaining easy representations for chordal graphs and considered several updating questions. The question that is important for our applications asks, given an edge, whether the represented graph remains chordal if the specified edge is removed. This question can be answered in time $\mathcal{O}(n)$ [44], which sums up to $\mathcal{O}(nm)$ time.

**Theorem 8.18 (Blair, Heggernes and Telle, [6])**
*There is an algorithm that decides the min-Tri membership problem for arbitrary graphs, i.e., given a pair $(G, H)$ of graphs, whether $H$ is a minimal triangulation of $G$, in time $\mathcal{O}(f(m + f))$ where $m$ denotes the number of edges in $G$ and $f$ denotes the number of edges in $H$ that are not edges in $G$.*

**Theorem 8.19 (Dahlhaus, [23])**
*There is an algorithm that decides the min-Tri membership problem for arbitrary graphs, i.e., given a pair $(G, H)$ of graphs, whether $H$ is a minimal triangulation of $G$, in time $\mathcal{O}(nm)$ where $m$ and $n$ denote the numbers of edges and vertices of $G$, respectively.*

**Theorem 8.20 (Ibarra, [44])**
*There is an algorithm that decides the min-Tri membership problem for arbitrary graphs, i.e., given a pair $(G, H)$ of graphs, whether $H$ is a minimal triangulation of $G$, in time $\mathcal{O}(fn + m)$ where $n$ and $m$ denote the numbers of vertices and edges of $H$, respectively, and $f$ denotes the number of edges of $H$ that are not edges in $G$.*

Our algorithms for min-Tri membership problems have far better running times. They profit from being applied to only restricted classes of graphs. Nevertheless, correctness and running time in the cases of arbitrary graphs as well as restricted graph classes rely on appropriate characterisations of minimal triangulations. And a considerable part of each chapter is dedicated to deducing and proving such characterisation theorems.

## 8.5 Width parameters

Minimal triangulations are interesting objects in their own right, and the study of minimal triangulations and related computational aspects created a rich field in graph theory and algorithms. But their influence reaches further parts of graph theory. So, they also appear in connection with a number of fundamental and well-studied graph notions. That is also why each of the following chapters in this part concludes with a section about applications of the obtained results. We are interested in the three graph parameters treewidth, minimum fill-in and bandwidth.

The treewidth of a graph is a measure for its “treelikeness”. It was introduced by Robertson and Seymour in their work about graph minors [74]. For defining the treewidth of a graph, we use a different approach. For a graph $G$ and a vertex $x$ that does not appear in $G$, $G{+}x$ denotes the graph that emerges from $G$ by adding $x$ as new vertex. The neighbourhood of $x$ must be defined separately. In some cases, $G$ is an induced subgraph of another graph. Then, the neighbourhood of $x$ is clearly determined by context. In fact, if $G$ is a subgraph of $G'$ induced by $V \subset V(G')$ and $x \in V(G') \setminus V$, then $G{+}x$ means the subgraph of $G'$ induced by $V \cup \{x\}$.

**Definition 28** *Let $k \geq 1$. A **k-tree** is a graph that can be obtained inductively by applying the following two operations:*

*(1) the clique on $k+1$ vertices is a $k$-tree*

*(2) let $G$ be a $k$-tree, and let $C$ be a clique of size $k$ in $G$; let $u$ be a vertex that does not appear in $G$; then $G{+}u$ with $u$ adjacent with exactly the vertices in $C$ is a $k$-tree.*

To get an idea of $k$-trees, note that 1-trees are just the usual trees, i.e., the connected graphs containing no cycles. By *partial $k$-trees* we mean subgraphs of $k$-trees; these subgraphs can be obtained by vertex as well as edge deletions.

**Definition 29** *Let $G = (V, E)$ be a graph. The **treewidth** of $G$, denoted as $\mathrm{tw}(G)$, is the smallest number $k$ such that $G$ is a partial $k$-tree.*

A proof of the equivalence of our definition of treewidth with the original definition introduced by Robertson and Seymour can be found in [55]. The following—very useful—characterisation of the treewidth of a graph is an immediate consequence, and it links treewidth and minimal triangulations.

**Lemma 8.21** *Let $G = (V, E)$ be a graph. Then,*

$$\mathrm{tw}(G) = \min\{\omega(H) : H \text{ is a minimal triangulation of } G\} - 1\,.$$

With the treewidth, we also associate a decision problem called TREEWIDTH, that is the set of pairs $(G, k)$ where $G$ is a graph and $\mathrm{tw}(G) \leq k$.

Another graph parameter that is closely related to minimal triangulations is the minimum fill-in.

**Definition 30** *Let $G = (V, E)$ be a graph. The **minimum fill-in** of $G$, denoted as $\mathrm{mfi}(G)$, is the least number of edges that have to be added to $G$ to obtain a chordal graph.*

In other words, the minimum fill-in gives the smallest number of additional edges among all triangulations of a graph. This parameter and triangulations with this number of additional edges turned out important in connection with numerical solutions of linear equation systems by Gaussian elimination (also see [75]). The following alternative definition of minimum fill-in is obvious.

**Lemma 8.22** *Let $G = (V, E)$ be a graph. Then,*

$$\text{mfi}(G) = \min\{|E(H)| - |E(G)| : H \text{ is a minimal triangulation of } G\}\,.$$

Analogous to treewidth, we can associate a decision problem with minimum fill-in. By Minimum Fill-in, we denote the set of pairs $(G, k)$ where $G$ is a graph and $\text{mfi}(G) \leq k$.

The third graph parameter, that we wish to consider, is motivated by an optimisation problem from linear algebra. Given a quadratic matrix $M$, is there a permutation matrix $P$ such that all non-zero entries in $P \cdot M \cdot P^{-1}$ appear only in a small area around the main diagonal? For graphs, we have the following definition.

**Definition 31** *Let $G = (V, E)$ be a graph on $n$ vertices. Let $\lambda : V \to \{1, \ldots, n\}$ be a bijective mapping; $\lambda$ is called a **layout** for $G$. The **width** of $\lambda$, denoted as $\text{b}(G, \lambda)$, is defined as*

$$\text{b}(G, \lambda) =_{\text{def}} \max\{|\lambda(u) - \lambda(v)| : uv \in E\}\,.$$

*The **bandwidth** of $G$, denoted as $\text{bw}(G)$, is the smallest width over all layouts for $G$, i.e.,*

$$\text{bw}(G) =_{\text{def}} \min\{\text{b}(G, \lambda) : \lambda \text{ is a layout for } G\}\,.$$

It is interesting to note that there is an alternative definition of the bandwidth of a graph using special triangulations. This connection will be presented in Chapter 11. Clearly, we can define a decision problem for the bandwidth. By Bandwidth, we denote the set of pairs $(G, k)$ where $G$ is a graph and $\text{bw}(G) \leq k$.

All three decision problems are NP-complete, even when they are restricted to small graph classes. A graph is *co-bipartite*, if its vertex set can be partitioned into two cliques. Co-bipartite graphs are the complements of bipartite graphs.

**Theorem 8.23 (Arnborg, Corneil and Proskurowski, [4])**
*The decision problem* Treewidth *is* NP*-complete for co-bipartite graphs.*

**Theorem 8.24 (Yannakakis, [88])**
*The decision problem* Minimum Fill-in *is* NP*-complete for co-bipartite graphs.*

**Theorem 8.25 (Kloks, Kratsch and Müller, [50])**
*The decision problem* Bandwidth *is* NP*-complete for co-bipartite graphs.*

# Chapter 9
# $2K_2$-free graphs

Graph classes can be defined in a number of different ways. The simplest and most inefficient manner is to select graphs one by one. Mostly, however, graphs are selected by special properties, more precisely, by structural properties. These selection rules may be of any kind. For instance, bipartite graphs are 2-colourable graphs, i.e., those graphs that admit colourings using at most two colours. As another example, chordal graphs are the graphs whose induced chordless cycles have lengths at most 3. This is a structural property. Alternatively, we can say that chordal graphs are exactly the graphs that do not contain a chordless cycle of length at least 4 as induced subgraph. This is a selection rule by giving a set of *forbidden induced subgraphs.* So, we distinguish graph classes according to their definitions that are based on

(1) structural properties

(2) forbidden induced subgraphs.

Formally, we say that a graph class $\mathcal{C}$ can be defined by forbidden induced subgraphs, if there is a set $\mathcal{H}$ of graphs such that graph $G$ belongs to class $\mathcal{C}$ if and only if $G$ does not contain any graph from $\mathcal{H}$ as induced subgraph. In that case, we also say that the graphs from $\mathcal{C}$ are *$\mathcal{H}$-free.* If $\mathcal{H}$ contains only one graph, we also omit braces to improve readability.

Many graph classes can be defined by forbidding induced subgraphs; mostly, an infinite set of forbidden subgraphs is necessary, as we have seen for chordal graphs: the class of chordal graphs is equal to the class of $\{C_4, C_5, C_6, \ldots\}$-free graphs. If graph classes are defined by forbidden induced subgraphs, it is most natural to consider finite sets of forbidden subgraphs. Among finite sets, small sets, and herein singleton sets, are of high interest.

Cographs have a lot of characterisations. One of them is by forbidden induced subgraph: cographs are exactly those graphs that do not contain the chordless path on four vertices, $P_4$, as induced subgraph. Cographs have many interesting properties, e.g., an efficient tree representation, which is of great importance for the design of algorithms. The question appeared whether some of these interesting algorithmic properties hold for graph classes that do not contain other chordless paths, and $P_5$-free graphs got into the focus of research. In the last years, the study of graph classes defined by one or few forbidden subgraphs became more and more popular. In this sequence, the $2K_2$, which is the complement of the chordless cycle on four vertices, $C_4$, fits well.

## 9.1 Getting to know $2K_2$-free graphs

The $2K_2$ is the graph on four vertices that contains exactly two edges, and the edges are non-adjacent. So, the $2K_2$ is the disjoint union of two connected graphs on two vertices. The $2K_2$ is the complement of the chordless cycle on four vertices, $C_4$. A copy of the $2K_2$ can be found in Figure 2.

**Definition 32** *A graph is $2K_2$-**free** if and only if it does not contain the $2K_2$ as induced subgraph.*

**Fact 9.1** *$2K_2$-free graphs are $P_5$-free.*

**Proof:** Let $G = (V, E)$ be a graph containing a $P_5$, and let it be induced by $\{u, v, w, x, z\} \subseteq V$ such that $(u, v, w, x, z)$ is a path in $G$. Then, $u$ is a neighbour of neither $x$ nor $z$, and $v$ is a neighbour of neither $x$ nor $z$. Hence, $\{u, v, x, z\}$ induce a $2K_2$ in $G$.

■

It is interesting that $2K_2$-free graphs allow characterisations via structural properties. This fact may support the idea that $2K_2$-free graphs are more "natural" than they seem at first glance. To formulate the statement we need two basic definitions. Let $G = (V, E)$ be a graph. The diameter of $G$, $\mathrm{diam}(G)$, is the length of the longest distance between two vertices of $G$; if $G$ is disconnected then $\mathrm{diam}(G) = \infty$. The *linegraph* $L(G)$ of $G$ contains a vertex for every edge of $G$, and two vertices of $L(G)$ are adjacent if they correspond to adjacent edges in $G$.

**Fact 9.2** *Let $G = (V, E)$ be a graph. The following statements are equivalent:*

*(1)* *$G$ is $2K_2$-free*

*(2)* *for every set $S \subseteq V$, $G \setminus S$ contains at most one connected component with two vertices*

*(3)* $\mathrm{diam}(L(G)) \leq 2$ .

**Proof:** We show that statement (1) is equivalent with statements (2) and (3).

(1) $\Leftrightarrow$ (2)
Let $S \subseteq V$, and let $G \setminus S$ contain two connected components with two vertices. Selecting a pair of adjacent vertices in every connected component gives a $2K_2$ as induced subgraph. If there is $S \subset V$ such that $V \setminus S$ induces a $2K_2$ in $G$, $G \setminus S$ contains two connected components with two vertices.

(1) $\Leftrightarrow$ (3)
Let $G$ be $2K_2$-free, and let $e_1, e_2 \in E$ be two non-adjacent edges. Since $G$ is $2K_2$-free, there must be an edge $e \in E$ that is adjacent with $e_1$ and $e_2$, which gives a path of length 2 between $e_1$ and $e_2$ in $L(G)$. Conversely, let $\{u, v, x, z\}$ induce a $2K_2$ in $G$ where $uv, xz \in E$. Then, $uv$ and $xz$ share no neighbour in $L(G)$, hence $\mathrm{diam}(L(G)) > 2$.

■

We want to relate the class of $2K_2$-free graphs to other graph classes and determine their relationships. We already mentioned the superclass of $P_5$-free graphs. Further graph classes that are of interest appear in connection with chordal graphs. The complements of chordal graphs are *co-chordal.*

**Definition 33** *Let $G = (V, E)$ be a graph. We call $G$ a* ***split graph*** *if and only if the vertex set of $G$ can be partitioned into an independent set $U$ and a clique $C$. The partition $\langle U, C\rangle$ is called a* ***split partition*** *of $G$.*

**Definition 34** *A graph is* ***weakly chordal*** *if and only if it and its complement do not contain a chordless cycle on more than four vertices as induced subgraphs.*

A forbidden subgraph characterisation of weakly chordal graphs is immediate, and this set is infinite. In case of split graphs, a similar characterisation is not obvious, but it turns out that split graphs are special chordal graphs. Even a characterisation that uses only a finite set of forbidden induced subgraphs is known.

**Theorem 9.3 (Földes and Hammer, [27])**
*The following statements are equivalent for a graph $G$:*

*(1) $G$ is a split graph*

*(2) $G$ is chordal and co-chordal*

*(3) $G$ is $\{2K_2, C_4, C_5\}$-free.*

Weakly chordal graphs are closed under complementation; this follows immediately from the definition. It is not difficult to see that chordal graphs are weakly chordal; hence, co-chordal graphs are also weakly chordal. We can relate the classes of $2K_2$-free, co-chordal and weakly chordal graphs.

**Fact 9.4** *A graph $G$ is $2K_2$-free and weakly chordal if and only if $G$ is co-chordal.*

**Proof:** If $G$ is $2K_2$-free, the complement of $G$ cannot contain the chordless cycle on four vertices as induced subgraph. If $G$ is weakly chordal, the complement of $G$ cannot contain a chordless cycle on at least five vertices as induced subgraph. Hence, if $G$ is $2K_2$-free and weakly chordal, the complement of $G$ is chordal and $G$ is co-chordal.

Conversely, if $G$ is not $2K_2$-free, the complement of $G$ contains a $C_4$. If $G$ is not weakly chordal, the complement of $G$ contains a chordless cycle on at least five vertices as induced subgraph. So, if $G$ is not $2K_2$-free or not weakly chordal, the complement of $G$ is not chordal, and $G$ is not co-chordal.

■

Using a forbidden subgraph notation, Fact 9.4 can be restated as follows:

$$\{2K_2\}\text{-free} \cap \{C_5, \overline{C_5}, C_6, \overline{C_6}, \ldots\}\text{-free} = \{\overline{C_4}, \overline{C_5}, \ldots\}\text{-free}\,.$$

Since we know that $2K_2$ and $\overline{C_4}$ denote isomorphic graph as well as $C_5$ and $\overline{C_5}$, and since $C_6$, $C_7$, $\ldots$ all contain the $2K_2$ as induced subgraph, i.e., a $2K_2$-free graph cannot contain $C_6$, $C_7$, $\ldots$ as induced subgraphs, the equivalence follows easily.

**Corollary 9.5** *A graph is $C_4$-free and weakly chordal if and only if it is chordal.*

**Proof:** It suffices to note that a graph is $C_4$-free and weakly chordal if and only if its complement is $2K_2$-free and weakly chordal. The statement follows from Fact 9.4.

■

Figure 17 summarises the results of Theorem 9.3, Fact 9.4 and Corollary 9.5. Note that the split graphs are contained in every mentioned graph class.

## 9.2 Characterisations of minimal triangulations of $2K_2$-free graphs

For a graph $G$, a *minimal fill-in* for $G$ is a minimal set of edges that are added to $G$ to make $G$ chordal, i.e., to obtain a minimal triangulation of $G$. So, adding a minimal fill-in can be considered an operation on graphs. In this section, we will answer the question in which way minimal fill-in affects the structural properties of $2K_2$-free graphs. In other words: what are the minimal triangulations of $2K_2$-free graphs, and can they be characterised non-trivially? We will see that $2K_2$-freeness is preserved by minimal fill-in.

**Theorem 9.6** *A graph is $2K_2$-free if and only if all its minimal triangulations are $2K_2$-free.*

**Proof:** We prove the statement by showing two implications.

[A graph containing a $2K_2$ has a minimal triangulation that contains a $2K_2$.]
Let $G = (V, E)$ be a graph, and let $U \subseteq V$ induce a $2K_2$ in $G$. Let $H' =_{\text{def}} (V, E \cup F)$ where $F =_{\text{def}} \{uv \notin E : \{u, v\} \not\subseteq U\}$. Since $H'$ is chordal and a spanning supergraph of $G$, $H'$ is a triangulation of $G$, and $U$ induces a $2K_2$ in $H'$. In every graph $G'$ where $G \subseteq G' \subseteq H'$, $U$ induces a $2K_2$. Hence, there is a minimal triangulation $H \subseteq H'$ of $G$ that contains a $2K_2$.

[A $2K_2$-free graph cannot have a minimal triangulation that contains a $2K_2$.]
Let $G = (V, E)$ be a graph, and let $H = G \cup F$, $F \cap E = \emptyset$, be a minimal triangulation of $G$. We will show that, if $H$ contains a $2K_2$, then $G$ contains a $2K_2$. By contraposition, it follows that $2K_2$-free graphs only have $2K_2$-free minimal triangulations.

Let $\{u, v, x, z\} \subseteq V$ induce a $2K_2$ in $H$ where $uv, xz \in E(H)$. We distinguish three cases. If $uv, xz \in E$, $\{u, v, x, z\}$ induces a $2K_2$ in $G$, and the proof is finished. Let $uv \in F$ and $xz \in E$. If there is a neighbour $w$ of $u$ that is adjacent with neither $x$ nor $z$ in $G$, $\{u, w, x, z\}$ induces a $2K_2$ in $G$. Otherwise, i.e., every neighbour of $u$ in $G$ is neighbour of $x$ or $z$ in $G$, $N_G(u)$ is a clique in $H$ (two non-adjacent neighbours of $u$ in $H$ would be contained in a chordless cycle with $u$ and $x$ or $z$). Since $u$ is not isolated in $G$, $u$ has a neighbour in $G$. Due to Lemma 8.17, however, $u$ cannot

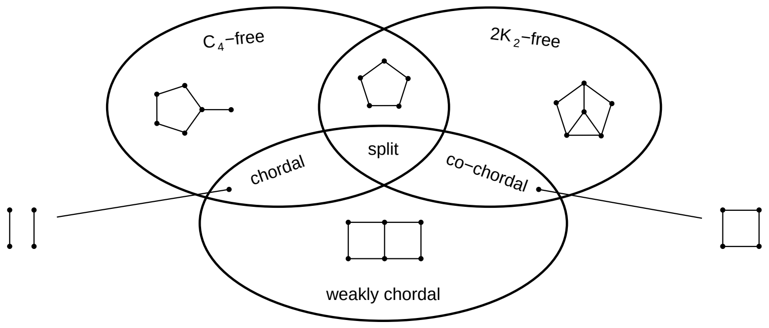


**Figure 17** A Venn diagram relating the class of $2K_2$-free graphs to well-known other graph classes.

be adjacent with $v$ in $H$, so that the situation $uv \in F$ and $xz \in E$ is not possible. Finally, let $uv, xz \in F$, i.e., $\{u, v, x, z\}$ is an independent set in $G$. Note that not every neighbour of $u$ in $G$ is a neighbour of $x$ or $z$ in $H$, since otherwise, $N_G(u)$ would be a clique in $H$, and $u$ could not be adjacent with $v$ in $H$ due to Lemma 8.17; let $w \in N_G(u)$ such that $wx, wz \notin E(H)$. Then, $\{u, w, x, z\}$ induces a $2K_2$ in $H$ with an edge already in $G$, and we can apply the second case.

■

**Corollary 9.7** *A graph is $2K_2$-free if and only if all its minimal triangulations are split graphs.*

**Proof:** Triangulations of graphs are chordal, and chordal $2K_2$-free graphs are split graphs due to Theorem 9.3. The statement then follows from Theorem 9.6.

■

From the first characterisation of minimal triangulations of $2K_2$-free graphs we can derive another characteristion. This second characterisation provides a means not only to characterise $2K_2$-free graphs similarly to the statement of Theorem 9.6 but also to characterise the minimal triangulations of a given graph. We will show that there is a strong correspondence between minimal triangulations of $2K_2$-free graphs and maximal independent sets.

Let $G = (V, E)$ be a graph. If $G$ contains at most one pair of adjacent vertices, i.e., at most one edge, we say that $G$ is *nearly blank*. For our characterisation, we define a set $\mathcal{U}_G$ of independent sets of $G$ in the following way. For every $U \subseteq V$, $U \in \mathcal{U}_G$ if and only if $U$ satisfies the three conditions:

(U1) $U$ is a maximal independent set of $G$

(U2) $\{a \in V : d_G(a) \leq 1\} \subseteq U$

(U3) if there is a vertex $b \in V \setminus U$ satisfying $|N_G(b) \cap U| = 1$ and $V \setminus U \nsubseteq N_G[b]$ then there is no vertex $a \in U$ such that $N_G(a) = V \setminus U$.

We observe that no independent set of a nearly blank graph with one edge can satisfy conditions (U1) and (U2) simultaneously. Since our second characterisation of minimal triangulations of $2K_2$-free graphs is based on the independent sets in $\mathcal{U}_G$, this may pose problems. However, nearly blank graphs are easily identifiable, and since they are chordal we simply exclude them from the following considerations. Hence, the graphs to be regarded subsequently do not contain adjacent vertices of degree 1.

**Lemma 9.8** *Let $G = (V, E)$ be a $2K_2$-free graph that is not nearly blank, and let $U \in \mathcal{U}_G$. Then, $H_U =_{\text{def}} G \cup F_U$ where $F_U =_{\text{def}} \{uv \notin E : u, v \in V \setminus U\}$ is a minimal triangulation of $G$.*

**Proof:** Of course, $H_U$ is a split graph: $U$ is an independent set and $V \setminus U$ is a clique in $H_U$. Thus, $H_U$ is chordal and a triangulation of $G$. Due to Theorem 8.14, we will show that every edge in $F_U$ is unique chord in a cycle of length 4 in $H_U$. For every vertex in $V \setminus U$, there is a neighbour in $U$ due to the maximality of $U$. If two vertices $u, v \in V \setminus U$ each have a neighbour in $U$ neither of which is a common neighbour of $u$ and $v$, then $uv \in E$. Let $uv \in F_U$. We assume $N_G(u) \cap U \subseteq N_G(v) \cap U$. If $u$ has two neighbours $x, z \in U$ then they are also neighbours of $v$, and $uv$ is unique chord in the cycle $(x, u, z, v)$. Note that $V \setminus U \not\subseteq N_G[u]$. If $u$ has exactly one neighbour $x$ in $U$ then, by condition (U3) for $U$ being in $\mathcal{U}_G$, there is a vertex $w \in V \setminus U$ that is not adjacent with $x$. Then, $w$ is adjacent with $u$ and $v$ and non-adjacent with $x$ in $H_U$, and $uv$ is unique chord in the cycle $(u, x, v, w)$ in $H_U$.

■

The following lemma strongly depends on Theorem 9.6. In fact, we assume that the vertex set of a minimal triangulation of a $2K_2$-free graph can be partitioned into a clique and an independent set.

**Lemma 9.9** *Let $G = (V, E)$ be a $2K_2$-free graph that is not nearly blank. Let $H = G \cup F$, $F \cap E = \emptyset$, be a minimal triangulation of $G$. Let $\langle U, V \setminus U\rangle$ be a partition of $V$ into a set $U$ that is maximally independent in $H$ and a clique $V \setminus U$ in $H$. Then, $U \in \mathcal{U}_G$.*

**Proof:** Due to Theorem 9.6, $H$ is a split graph, and the vertex set of split graphs can be partitioned into an independent set and a clique by definition. Hence, the partition $\langle U, V \setminus U\rangle$ is well-defined. Of course, $N_H(u) = N_G(u)$ for every vertex $u \in U$, so that $U$ is a maximal independent set in $G$, too. An isolated vertex of $G$ must be an isolated vertex of $H$. A vertex of degree 1 in $G$ has degree 1 in $H$. Hence, all vertices of degree less than 2 must be contained in $U$. Remember that $G$ does not contain adjacent vertices of degree 1. So, $U$ fulfills conditions (U1) and (U2) for being member of $\mathcal{U}_G$. Finally, assume there are $u, v \in V \setminus U$ such that $u$ has exactly one neighbour in $U$ and $u$ and $v$ are non-adjacent in $G$. Then, $uv \in F$, and $uv$ is unique chord in a cycle of length 4 due to Theorem 8.14. There are two non-adjacent common neighbours of $u$ and $v$ in $H$, and one of them is not contained in $U$. Hence, $U$ fulfills condition (U3), thus $U \in \mathcal{U}_G$.

■

The two last lemmata constitute our second characterisation of minimal triangulations of $2K_2$-free graphs.

**Theorem 9.10** *Let $G = (V, E)$ be a $2K_2$-free graph that is not nearly blank, and let $H$ be a graph on vertex set $V$. Then, $H$ is a minimal triangulation of $G$ if and only if there is $U \in \mathcal{U}_G$ such that $H = H_U$.*

**Proof:** If $H$ is a minimal triangulation of $G$, then there is $U \in \mathcal{U}_G$ such that $H = H_U$ due to Lemma 9.9. If $H = H_U$ for some $U \in \mathcal{U}_G$, $H$ is a minimal triangulation of $G$ due to Lemma 9.8.

■

**Corollary 9.11** *Let $G = (V, E)$ be a graph that is not nearly blank. $G$ is $2K_2$-free if and only if for every minimal triangulation $H$ of $G$ there is $U \in \mathcal{U}_G$ such that $H = H_U$.*

**Proof:** If $G$ is $2K_2$-free, the statement holds due to Theorem 9.10. If $G$ is not $2K_2$-free, there is a minimal triangulation $H$ of $G$ that is not $2K_2$-free due to Theorem 9.6. Then, there is no independent set $U$ of $G$ such that $H = H_U$, since $H_U$ is always a split graph, i.e., a chordal graph not containing a $2K_2$.

■

## 9.3 Solving the min-Tri membership problem

The membership problem that we wish to solve here is defined as follows: given a pair $(G, H)$ of graphs, where $G$ is a $2K_2$-free graph, is $H$ a minimal triangulation of $G$? We will call this problem the *min-Tri membership problem for $2K_2$-free graphs.* A variant of this problem is called the *promise min-Tri membership problem for $2K_2$-free graphs* where we trust that the input graph $G$ is $2K_2$-free. In fact, the difference between both problems is that we also have to check whether $G$ is $2K_2$-free in the former case, whereas in the latter case, this can be assumed. So, recognition of $2K_2$-free graphs becomes part of the decision algorithm.

First, we will give an algorithm for solving the promise min-Tri membership problem for $2K_2$-free graphs that runs in two phases: checking whether $H$ is a split graph, and then, checking whether the vertex set of $H$ can be partitioned according to Lemma 9.9. The following result provides a quite nice recognition algorithm for split graphs.

**Theorem 9.12 (Hammer and Simeone, in [35])**
*Let $G = (V, E)$ be a graph on $n$ vertices. $G$ is a split graph if and only if there is $m \leq n$ such that $\{u \in V : d_G(u) \leq m\}$ is an independent set in $G$ and $\{u \in V : d_G(u) > m\}$ is a clique in $G$. The separating number $m$ can be computed in linear time.*

**Proof:** If there is $m \in \{0, \ldots, n\}$ such that $\langle U, C \rangle$ is a split partition of $G$ where $U =_{\text{def}} \{u \in V : d_G(u) \leq m\}$ and $C =_{\text{def}} \{u \in V : d_G(u) > m\}$, then $G$ is a split graph. If $G$ is a split graph, there exists $U \subseteq V$ such that $U$ is an independent set in $G$ and $C =_{\text{def}} V \setminus U$ is a clique in $G$. Then, $d_G(v) \geq |C| - 1$ for every $v \in C$ and $d_G(u) \leq |C|$ for every $u \in U$. If there is $w \in C$ such that $d_G(w) = |C| - 1$, then $v$ does not have any neighbour in $U$, and $\langle U \cup \{w\}, C \setminus \{w\} \rangle$ is also a split partition of $G$, and every vertex in $C \setminus \{w\}$ has a neighbour in $U \cup \{w\}$. Thus, there is $U' \subseteq V$ such that $\langle U', C' \rangle$, where $C' =_{\text{def}} V \setminus U'$, is a split partition of $G$, and every vertex in $C'$ has a neighbour in $U'$. Hence, $d_G(u) \leq |C'|$ for every $u \in U'$ and $d_G(v) \geq |C'|$ for every $v \in C'$. If there are two vertices $u_1, u_2 \in U'$ such that $d_G(u_1) = d_G(u_2) = |C'|$, then $d_G(v) \geq |C'| + 1$ for every $v \in C'$, and $m =_{\text{def}} |C'|$ is the separating number. If there is no such vertex, $d_G(u) \leq |C'| - 1$ for every $u \in U'$, and $m =_{\text{def}} |C'| - 1$ is the separating number. If there is exactly one vertex $u \in U'$ such that $d_G(u) = |C'|$, then $C' \cup \{u\}$ is a clique in $G$, and $m =_{\text{def}} |C'| - 1$ is the separating number.

For computing number $m$, let $d_1, \ldots, d_n$ be the degrees of the vertices of $G$ where $d_1 \leq \cdots \leq d_n$. It follows from the discussion above that $m = \max\{d_i : d_i < n - i\}$ can be chosen. This maximum can be determined in linear time.

■

**Theorem 9.13** *Algorithm* `MSP_minTri_cosquarefree`*, given in Figure 18, solves the promise min-Tri membership problem for $2K_2$-free graphs in linear time.*

**Proof:** Let $(G, H)$ be an instance of the promise min-Tri membership problem for $2K_2$-free graphs. In particular, $G$ is a $2K_2$-free graph. In linear time, the following can be tested: whether $G$ and $H$ are defined on the same vertex set, whether $G$ contains at most one edge and whether $G = H$, whether $H$ is a split graph. If all this is the case, a partition $\langle U, V(H) \setminus U \rangle$ of the vertex set of $H$ can be computed due to Theorem 9.12. It holds that $U$ is a maximal independent set of $H$ if and only if every vertex in $V(H) \setminus U$ has a neighbour in $U$. Hence, $U$ can be made maximal by adding at most one vertex from $V(H) \setminus U$. Due to Lemma 9.9 and Theorem 9.10, $H$ is a minimal triangulation of $G$ if and only if $U \in \mathcal{U}_G$. Conditions (U1) and (U2) for being in $\mathcal{U}_G$ can be verified in linear time. Condition (U3) can be verified as follows: determine the number of vertices in $U$ whose neighbourhoods in $G$ are $V(G) \setminus U$. If there is exactly one such vertex $w$, no vertex in $V(G) \setminus U$ with $w$ the only neighbour in $U$ shall have degree $|V \setminus U|$ in $G$. This condition is linear-time verifiable. Hence, `MSP_minTri_cosquarefree` is a linear-time algorithm.

■

It is certainly clear that algorithm `MSP_minTri_cosquarefree` does not always correctly work on graphs that contain a $2K_2$. As an example, consider "special graph 1" and "special graph 2" of Figure 2. Special graph 1 contains a $2K_2$, and special graph 2 is a triangulation of special graph 1. Since special graph 1 is chordal, special graph 2 is not a minimal triangulation of special graph 1, but falsely declared a minimal triangulation by `MSP_minTri_cosquarefree`. So, for solving the min-Tri membership problem for $2K_2$-free graphs, we have to verify that the first input graph is $2K_2$-free. The currently best known recognition algorithm is based on matrix multiplication. Let $\alpha$ be smallest such that the product of two matrices can be computed in time $\mathcal{O}(n^\alpha)$; it holds that $\alpha < 2.376$ [18].

**Theorem 9.14 (Kloks, Kratsch and Müller, [51])**
*$2K_2$-free graphs can be recognized in time $\mathcal{O}(n^\alpha + m^{\frac{1}{2}(\alpha+1)})$.*

**Corollary 9.15** *The min-Tri membership problem for $2K_2$-free graphs can be solved in time $\mathcal{O}(n^\alpha + m^{\frac{1}{2}(\alpha+1)})$.*

```
MSP_minTri_cosquarefree($G$, $H$) returns Boolean:
begin
   if (($V(G) \neq V(H)$) or ($E(G) \not\subseteq E(H)$)) then
      return false
   end if;
   if nearly_blank($G$) then
      return ($G = H$)
   end if;
   if not(split_graph($G$)) then
      return false
   end if;
   compute a maximal independent set $U$ of $H$
      such that $V(G) \setminus U$ is a clique in $H$;
   return ($U \in \mathcal{U}_G$)
end.
```

**Figure 18** A linear-time algorithm for solving the min-Tri membership problem for $2K_2$-free graphs. For correctness, it is assumed that input graph $G$ is $2K_2$-free.

**Proof:** Let $(G, H)$ be a pair of graphs. Due to Theorem 9.14, $G$ can be recognized as $2K_2$-free in the given time bound. If the test is positively answered, the algorithm `MSP_minTri_cosquarefree` decides whether $H$ is a minimal triangulation of $G$ in linear time due to Theorem 9.13. Since $\alpha \geq 1$ holds, the whole algorithm works in the stated time.

■

## 9.4 Algorithmic applications

Our second characterisation of minimal triangulations of $2K_2$-free graphs implies efficient algorithms for a number of minimal triangulations problems, in particular, for computing a minimal triangulation and for determining treewidth and minimum fill-in of $2K_2$-free graphs.

**Theorem 9.16** *There is a linear-time algorithm for computing a minimal triangulation of a $2K_2$-free graph, where the triangulation is represented by a split partition.*

**Proof:** Let $G = (V, E)$ be a $2K_2$-free graph. Due to Theorem 9.10, it suffices to compute an independent set from set $\mathcal{U}_G$. In linear time the degree of each vertex can be computed and the vertices can be ordered accordingly. If $G$ is nearly blank, $G$ is chodal. Let $u \in V$ have degree 1. Then, $\langle V \setminus \{u\}, \{u\}\rangle$ is an appropriate split partition. Otherwise, if $G$ is not nearly blank, compute a maximal independent set $U$ of $G$ as follows: choosing the vertices in order of their degrees starting with the smallest and put a vertex to the independent set whenever it is possible. This greedy algorithm is linear-time. We have to show that $U \in \mathcal{U}_G$:

(a) $U$ fulfills condition (U1) by construction.

(b) $U$ contains all vertices with degree 0. Suppose there is a vertex of degree 1 that is not contained in $U$. A neighbour must have been selected earlier, but this could only be a vertex of degree 1, which would imply that $G$ contains a connected component of exactly two vertices and which is not possible, since $G$ is not nearly blank and $2K_2$-free. Hence, $U$ fulfulls condition (U2).

(c) Suppose there is a vertex $u \in U$ that is adjacent with every vertex in $V \setminus U$. If there is a vertex $v$ in $V \setminus U$ such that $u$ is its only neighbour in $U$ and that is not adjacent with all vertices in $V \setminus U$, $v$ must have smaller degree than $u$. But the algorithm would have chosen $v$ before $u$. Hence, $U$ fulfills condition (U3).

Then, $U \in \mathcal{U}_G$, and $\langle U, V \setminus U\rangle$ is an appropriate split partition.

■

Theorem 9.10 shows that the number of minimal triangulations of a $2K_2$-free graph is bounded above by the number of its maximal independent sets. For obtaining efficient algorithms for computing treewidth and minimum fill-in of $2K_2$-free graphs, we will simply enumerate all maximal independent sets and explicitly consider every minimal triangulation. Efficiency of this algorithm is based on the fact that $2K_2$-free graphs have only a small number of maximal independent sets, and they can efficiently be generated. The algorithm is based on ideas that were independently used by Farber and Prisner for showing that a $2K_2$-free graph with $m$ edges has at most $m+1$ maximal independent sets [26], [71].

**Lemma 9.17** *There is an $\mathcal{O}(n+m^2)$-time algorithm that generates all maximal independent sets of a $2K_2$-free graph.*

**Proof:** We give here the proof of Farber and Prisner, that provide the algorithm. Let $G=(V,E)$ be a $2K_2$-free graph, and let $x_1,\ldots,x_n$ be the vertices of $G$. For the following considerations, we assume that $G$ does not contain isolated vertices; in linear time, they can be identified and deleted. Hence, $G$ is connected and $m \geq n-1$. We generate the maximal independent sets iteratively. For $i<n$, let the set of maximal independent sets of $G_i =_{\mathrm{def}} G[\{x_1,\ldots,x_i\}]$ be given. Let $U$ be a maximal independent set of $G_{i+1}=G[\{x_1,\ldots,x_{i+1}\}]$ containing $x_{i+1}$. Either $U\setminus\{x_{i+1}\}$ is a maximal independent set of $G_i$, or $U\setminus\{x_{i+1}\}$ is properly contained in a maximal independent set $U'$ of $G_i$. In the latter case, there is $u\in U'\cap N_G(x_{i+1})$ such that $U\setminus\{x_{i+1}\}=\{x_1,\ldots,x_i\}\setminus(N_G(x_{i+1})\cup N_G(u))$. That the right hand side defines an independent set is due to the $2K_2$-freeness of $G$, and the inclusion from right to left holds by maximality of $U$. So, if $G_i$ has $R_i$ maximal independent sets, $G_{i+1}$ has $R_{i+1}\leq R_i+d_G(x_{i+1})$ maximal independent sets. Hence, $R_j\leq m+1$ for all $j\in\{1,\ldots,n\}$.

For an efficient implementation of the above sketched generation algorithm, we store the independent sets in an $(m{+}1)\times n$ matrix, each row representing a maximal independent set. In time $\mathcal{O}(nm)$ it can be initialized with 0 in every position. For each vertex, we keep a list of descending indices of the independent sets in which this vertex appears. The index of an independent set is just the number of the corresponding row in the matrix. To find the independent sets of $G_i$ that can be extended by $x_{i+1}$ to be a maximal independent set in $G_{i+1}$, start with a descending list containing the indices of all already generated independent sets and exclude indices by running through the list of each neighbour of $x_{i+1}$. Since all lists are ordered, this takes time $\mathcal{O}(m\cdot d_G(x_{i+1}))$. For all independent sets kept in the final list, it remains to change a bit in the matrix, which adds time $\mathcal{O}(m)$. Now, consider the sets $U_{i+1}^j =_{\mathrm{def}} \{x_1,\ldots,x_i\}\setminus(N_G(x_{i+1})\cup N_G(x_j))$ for $x_j\in N_G(x_{i+1})$ and $j\leq i$. All these sets are independent sets, but none of them is maximal in $G_i$, hence $U_{i+1}^j\cup\{x_{i+1}\}$ where $x_j\in N_G(x_{i+1})$ and $j\leq i$ does not appear in the already generated list of maximal independent sets of $G_{i+1}$. It holds that $U_{i+1}^j=U_{i+1}^{j'}$ if and only if $N_{G_i}(x_j)\setminus N_G(x_{i+1})=N_{G_i}(x_{j'})\setminus N_G(x_{i+1})$. In linear time equal independent sets can be identified. So, it remains to find the maximal sets among the remaining of the sets $U_{i+1}^1,\ldots,U_{i+1}^i$. We verify for every $j\in\{1,\ldots,i\}$ whether each set contains a vertex from $N_G[x_j]$. Using adjacency lists, this takes time $\mathcal{O}(m\cdot d_G(x_{i+1}))$. Adding the found maximal independent sets among $U_{i+1}^1,\ldots,U_{i+1}^i$ to the list and updating the indices lists takes time $\mathcal{O}(n \cdot d_G(x_{i+1}))$, which adds up to $\mathcal{O}(m \cdot d_G(x_{i+1}))$ time for $x_{i+1}$ and $\mathcal{O}(n + m^2)$ for the whole described algorithm.

■

**Corollary 9.18** *For a $2K_2$-free graph $G, \mathcal{U}_G$ can be generated in time $\mathcal{O}(n + m^2)$. If $G$ has $m$ edges, $|\mathcal{U}_G| \leq m + 1$.*

**Proof:** Let $G = (V, E)$ be a $2K_2$-free graph. If $G$ is nearly blank, output the empty set. Otherwise, we start with the list of maximal independent sets of $G$ generated by the algorithm described in the proof of Lemma 9.17. For every independent set, conditions (U1–3) have to be verified. Conditions (U1) and (U2) are easily checkable. A procedure for verifying condition (U3) is given in the proof of Theorem 9.16, which results in a linear-time algorithm for every independent set. Since there are at most $m + 1$ maximal independent sets of $G$, we obtain a $\mathcal{O}(n + m^2)$-time algorithm.

■

**Theorem 9.19** *Treewidth and minimum fill-in of $2K_2$-free graphs can be computed in time $\mathcal{O}(n + m^2)$.*

**Proof:** Let $H$ be a split graph, and let $\langle U, C \rangle$ be a split partition of $H$. Then, $|C| \leq \omega(H) \leq |C| + 1$, since $C$ is a clique in $H$, and at most one vertex from $U$ can increase clique $C$. Hence, the clique number of a split graph can be determined in linear time from a split partition. Due to Corollary 9.18 and Theorem 9.10, in time $\mathcal{O}(n + m^2)$, the smallest clique number $k$ among all minimal triangulations of $G$ can be determined. The treewidth of $G$ then is $k - 1$.

In case of minimum fill-in, we observe that, for every $U \in \mathcal{U}_G$,

$$|E(H_U)| = \binom{|V \setminus U|}{2} + \sum_{u \in U} d_G(u) \,.$$

The smallest number among all these values can be determined in time $\mathcal{O}(n + m^2)$ and is equal to $\mathrm{mfi}(G) + |E(G)|$.

■

# Chapter 10
# Permutation graphs

Many interesting graph classes can be defined as intersection graphs of appropriate geometric objects. As we have seen, chordal graphs are the intersection graphs of families of subtrees of trees (Theorem 8.12), and trees can certainly be considered geometric objects. For permutation graphs, there is also a nice intersection model: permutation graphs are the intersection graphs of sets of line segments that have endpoints on two parallel lines. This model, called *permutation diagram*, provides all information about the represented graph in a concise form. It is often used for algorithmic purposes and the design of efficient algorithms in many cases.

In this chapter, we will prove a new characterisation of minimal triangulations of permutation graphs. This characterisation is based on the concept of potential maximal cliques, that turn out to be useful not only for general graphs. The notion of a potential maximal clique was introduced by Bouchitté and Todinca [13]. A potential maximal clique of a graph $G$ is a set of vertices that is a maximal clique in a minimal triangulation of $G$. The authors used these potential maximal cliques to solve the treewidth and minimum fill-in problems on weakly chordal graphs in polynomial time [13].

We will give an easy characterisation of potential maximal cliques of permutation graphs that is based on special parts of permutation diagrams. Given this characterisation, it seems almost natural to define the so-called *potential maximal cliques graph* of a permutation graph, that is an acyclic graph and whose maximal paths exactly represent the minimal triangulations of the graph. Using this representation, we can solve the min-Tri membership problem for permutation graphs, that asks, given a pair $(G, H)$ of graphs, where $G$ is a permutation graph, whether $H$ is a minimal triangulation of $G$.

As another highly interesting application of our efficient minimal triangulations representation by potential maximal cliques graphs, we can solve the treewidth and minimum fill-in problems on permutation graphs in linear time.

## 10.1 Getting to know AT-free and permutation graphs

AT-free graphs have been introduced by Lekkerkerker and Boland for characterising interval graphs as special chordal graphs [56]. For a long period of time, AT-free graphs have not been considered. But in the last decade of the twentieth century, the class of AT-free graphs became the centre of extensive studies, that were mainly initiated by the works of Corneil, Olariu, Stewart about structural properties of AT-free graphs [21].

**Definition 35** *Let $G = (V, E)$ be a graph. A triple of vertices $u, v, w$ of $G$ is an* ***asteroidal triple*** *of $G$ if and only if every pair of vertices of the set $\{u, v, w\}$ is connected by a path in $G$ that does not contain a neighbour of the third vertex. We say that $G$* ***contains*** *an asteroidal triple if and only if there are three vertices $u, v, w$ of $G$ such that $u, v, w$ is an asteroidal triple of $G$.*

Alternatively, we can say that a triple $u, v, w$ of vertices of a graph $G = (V, E)$ is an asteroidal triple of $G$ if and only if, for every $x \in \{u, v, w\}$, $\{u, v, w\} \setminus \{x\}$ is contained in one connected component of $G \setminus N_G(x)$. Examples for graphs that contain asteroidal triples are depicted in Figure 19.

**Definition 36** *Let $G = (V, E)$ be a graph. We say that $G$ is* ***AT-free*** *if and only if $G$ does not contain an asteroidal triple.*

An interesting and important class of graphs contained in the class of AT-free graphs is the class of *cocomparability graphs*. Let $G = (V, E)$ be a graph. An orientation of the edges of $G$ is an assignment $\alpha$ of orientations to every edge of $G$ such that, for every edge $uv$ of $G$, either $\alpha(uv) = (u, v)$ or $\alpha(uv) = (v, u)$. We say that $\alpha$ is a *transitive orientation* of $G$ if and only if, for every triple $u, v, w$ of vertices of $G$ where $uv \in E$ and $vw \in E$, holds:

$$\alpha(uv) = (u, v) \text{ and } \alpha(vw) = (v, w) \quad \Longrightarrow \quad uw \in E \text{ and } \alpha(uw) = (u, w)\,.$$

A *comparability graph* is a graph that has a transitive orientation. It is interesting to note that comparability graphs can be characterised by vertex orderings. It holds that a graph $G = (V, E)$ is a comparability graph if and only if there is a vertex ordering $\sigma$ for $G$ such that, for every triple $u, v, w$ of vertices of $G$, $u \prec_\sigma v \prec_\sigma w$, $uv \in E$ and $vw \in E$ implies $uw \in E$. Now, a graph is a *cocomparability graph*, if it is the complement of a comparability graph. Similarly, it holds that $G = (V, E)$ is a cocomparability graph if and only if there is a vertex ordering $\sigma$ for $G$ such that, for every triple $u, v, w$ of vertices of $G$, $u \prec_\sigma v \prec_\sigma w$, $uw \in E$ implies $uv \in E$ or $vw \in E$ [54]. Such a vertex ordering is called *cocomparability ordering*.

**Lemma 10.1 (Golumbic, Monma and Trotter, [36])**
*Cocomparability graphs are AT-free.*

**Proof:** Let $G = (V, E)$ be a cocomparability graph, and let $\sigma$ be a cocomparability ordering for $G$. Suppose $G$ contains an asteroidal triple, formed by the vertices $u, v, w$. We may assume that $u \prec_\sigma v \prec_\sigma w$. By definition, there is a $u, w$-path $P$ in $G$ avoiding the neighbourhood of $v$. Then, there are adjacent vertices $x, z$ on $P$ such that $x \prec_\sigma v \prec_\sigma z$. But since $\sigma$ is a cocomparability ordering for $G$, $x$ or $z$ is a neighbour of $v$, which contradicts $P$ being a $u, w$-path avoiding the neighbourhood of $v$, and $G$ is AT-free.

■

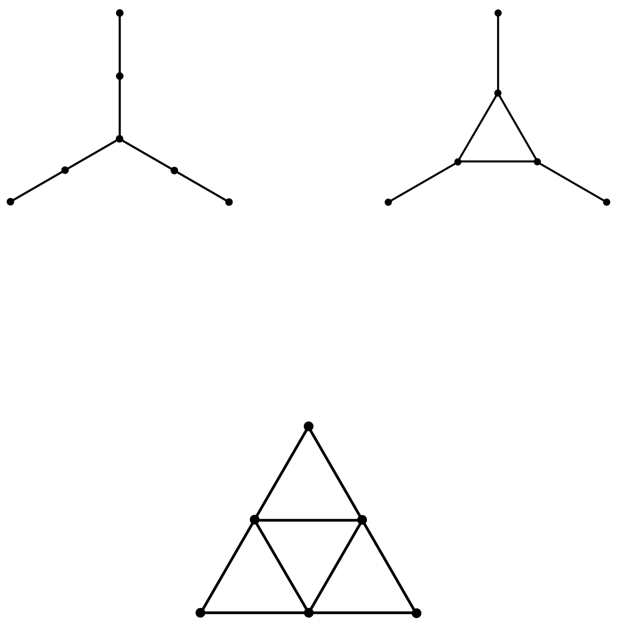


**Figure 19** Standard graphs containing asteroidal triples. In all three graphs, the three vertices with the smallest neighbourhoods form asteroidal triples.

Permutation graphs are cocomparability graphs. Let $n \geq 1$. A *permutation* over $\{1, \ldots, n\}$ is a bijective mapping $\pi$ from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$. A short description of $\pi$ is the sequence $\langle \pi(1), \ldots, \pi(n) \rangle$. By $\pi^{-1}$, we mean the inverse function of $\pi$, i.e., $\pi^{-1}(i) = j$ if and only if $\pi(j) = i$.

**Definition 37** *Let $n \geq 1$, and let $\pi$ be a permutation over $\{1, \ldots, n\}$. The **permutation graph defined by** $\pi$, denoted as $G(\pi)$, has vertex set $\{1, \ldots, n\}$, and two vertices $i$ and $j$, $i < j$, of $G(\pi)$ are adjacent if and only if $\pi(i) > \pi(j)$. A graph $G$ on $n$ vertices is a **permutation graph** if and only if there is a permutation $\pi$ over $\{1, \ldots, n\}$ such that $G = G(\pi)$. In this case, we also say that $G$ is a **permutation graph over** $\{1, \ldots, n\}$.*

The definition of permutation graphs that we use here is rather strict, since the class of permutation graphs is not closed under isomorphism. However, for our considerations, this circumstance can be ignored.

**Theorem 10.2 (McConnell and Spinrad, [60])**
*Let $G = (V, E)$ be a graph. If $G$ is isomorphic to a permutation graph, then a permutation $\pi$ can be computed in linear time such that $G \cong G(\pi)$.*

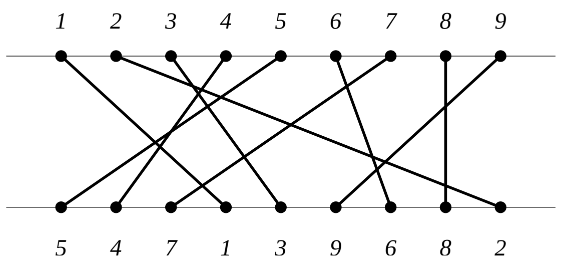
There is a nice representation of permutation graphs, that is closely related to permutation sequences. It is called *permutation diagram.* Let $G = G(\pi)$ be a permutation graph on $n$ vertices. On each of two horizontal lines, mark $n$ equidistant points. On the upper line, label the points from left to right with the numbers 1 through $n$, on the lower line, label the points from left to right with the numbers $\pi(1)$ through $\pi(n)$. Finally, join the points on the upper and lower line with the same labels by line segments. The result is the permutation diagram of $G$, and it will often be denoted as $\mathfrak{D}(\pi)$. An example is depicted in Figure 20. It holds that two vertices of $G$ are adjacent if and only if the corresponding line segments intersect. We will often speak of intersecting line segments, even of intersecting vertices instead of adjacent vertices.

**Lemma 10.3 (Pnueli, Lempel and Even, [70])**
*Permutation graphs are cocomparability graphs.*

**Proof:** Let $G = G(\pi) = (V, E)$ be a permutation graph over $\{1, \ldots, n\}$. We show that $\sigma =_{\text{def}} \langle \pi(1), \ldots, \pi(n) \rangle$ is a cocomparability ordering for $G$. Let $u, v, w$ be vertices of $G$ where $\pi^{-1}(u) < \pi^{-1}(v) < \pi^{-1}(w)$, i.e., $u \prec_\sigma v \prec_\sigma w$, and let $uw \in E$. By definition, $w < u$. Then, if $v < u$, $v$ and $u$ are adjacent, if $v > u$, $v$ and $w$ are adjacent. Hence, $\sigma$ is a cocomparability ordering for $G$ and $G$ is a cocomparability graph.

■

It can even be shown that permutation graphs are exactly the cocomparability graphs that are also comparability graphs [70].



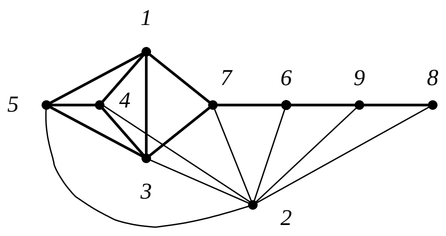


**Figure 20** The permutation graph $G(\langle 5,4,7,1,3,9,6,8,2\rangle)$ represented by a permutation diagram and by a usual "drawing".

The class of AT-free graphs also contains an important class of chordal graphs, namely the class of interval graphs.

**Definition 38** *A graph $G = (V, E)$ is an* ***interval graph*** *if and only if there is a 1-to-1 correspondence between the vertices of $G$ and a family $\mathfrak{I}$ of closed intervals of the real line such that two vertices of $G$ are adjacent if and only if the corresponding intervals have a non-empty intersection. The family of intervals is called an* ***interval model*** *for $G$, and we write $G(\mathfrak{I})$ to denote the graph defined by $\mathfrak{I}$.*

**Theorem 10.4 (Lekkerkerker and Boland, [56])**
*A graph is an interval graph if and only if it is chordal AT-free, i.e., chordal and AT-free.*

It is interesting to know that interval graphs can also be characterised by an appropriate vertex ordering, that is called interval ordering.

**Definition 39** *Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. We say that $\sigma$ is an* ***interval ordering*** *for $G$ if and only if, for every triple $u, v, w$ of vertices of $G$, holds:*

$$u \prec_\sigma v \prec_\sigma w \text{ and } uw \in E \quad \Longrightarrow \quad vw \in E\,.$$

It is immediately obvious that interval orderings are cocomparability orderings.

**Theorem 10.5 (Olariu, [66])**
*A graph is an interval graph if and only if it has an interval ordering.*

**Proof:** Let $G = (V, E)$ be an interval graph, and let $\mathfrak{I}$ be a family of closed intervals of the real line such that $G = G(\mathfrak{I})$. Additionally, let $\tau$ be a vertex ordering for $G$. For every vertex $x$ of $G$, let $I_x = [\ell(x), r(x)]$ be the interval of $\mathfrak{I}$ corresponding to $x$. We define a vertex ordering $\sigma$ of $G$ is follows. For every pair $u, v$ of vertices of $G$, let $u \prec_\sigma v$ if and only if $r(u) < r(v)$ or, if $r(u) = r(v)$, $u \prec_\tau v$. For showing that $\sigma$ is an interval ordering for $G$, let $u, v, w$ be vertices of $G$ where $u \prec_\sigma v \prec_\sigma w$ and $uw \in E$. By definition of $\sigma$, $r(u) \leq r(v) \leq r(w)$. Since $u$ and $w$ are adjacent in $G$, $I_u$ and $I_w$ intersect, i.e., $\ell(w) \leq r(u)$. Then, $\ell(w) \leq r(v)$, and $v$ and $w$ are adjacent in $G$.

For the converse, let $G = (V, E)$ be a graph, and let $\sigma$ be an interval ordering for $G$. We define a family $\mathfrak{I}$ of intervals of the real line as follows. For every vertex $x$ of $G$, let $a_x$ be the smallest number such that $\sigma(a_x)$ is contained in $N_G[x]$. Then, let $I_x =_{\text{def}} [a_x, \sigma^{-1}(x)]$, and let $\mathfrak{I} =_{\text{def}} \{I_x\}_{x \in V}$. It is clear that $a_x \leq \sigma^{-1}(x)$ for every vertex $x$ of $G$. We show that $G = G(\mathfrak{I})$ holds. Let $u, v$ be vertices of $G$, $u \prec_\sigma v$. First, let $I_u \cap I_v = \emptyset$. By definition of $\mathfrak{I}$, $\sigma^{-1}(u) < a_v$, which means that $u$ and $v$ are not adjacent in $G$. Second, let $I_u \cap I_v \neq \emptyset$. Then, $a_v \leq \sigma^{-1}(u)$, and $u$ and $v$ are adjacent in $G$ due to the definition of interval orderings. Hence, $\mathfrak{I}$ is an interval model for $G$.

■

Note that the proof of Theorem 10.5 provides a linear-time algorithm for computing an interval model for an interval graph from a given interval ordering. The converse is also true, if sorting can be done in linear time.

Another characterisation of interval graphs is given by a special property of its maximal cliques.

**Definition 40** *Let $G = (V, E)$ be a graph, and let $A_1, \ldots, A_r$ be the maximal cliques of $G$. We say that $G$ has a* ***consecutive clique arrangement*** *if and only if there is a permutation $\pi$ over $\{1, \ldots, r\}$ such that, for every vertex $x$ of $G$, if $x$ is vertex in $A_{\pi(i)}$ and $A_{\pi(j)}$ for $1 \leq i \leq j \leq r$ then $x$ is vertex in $A_{\pi(i)}, A_{\pi(i+1)}, \ldots, A_{\pi(j)}$. The sequence $\langle A_{\pi(1)}, \ldots, A_{\pi(r)} \rangle$ is a* ***consecutive clique arrangement*** *for $G$.*

**Theorem 10.6 (Gilmore and Hoffman, [33])**
*Let $G = (V, E)$ be a graph. Then, $G$ is an interval graph if and only if $G$ has a consecutive clique arrangement.*

**Proof:** Let $\langle A_1, \ldots, A_r \rangle$ be a consecutive clique arrangement for $G$. We define an interval model $\mathfrak{I}$ as follows. For every vertex $x$ of $G$, let

$$\begin{aligned} \ell(x) &=_{\text{def}} \min\{i : x \in A_i\} \\ r(x) &=_{\text{def}} \max\{i : x \in A_i\}, \end{aligned}$$

and $I_x =_{\text{def}} [\ell(x), r(x)]$. Note that there may be intervals containing only one element. For every pair $u, v$ of vertices of $G$, $uv \in E$ if and only if there is $i \in \{1, \ldots, r\}$ such that $u, v \in A_i$, if and only if $I_u \cap I_v \neq \emptyset$. Hence, $G = G(\mathfrak{I})$, where $\mathfrak{I} =_{\text{def}} \{I_x\}_{x \in V}$.

For the converse, let $G$ be an interval graph, and let $\mathfrak{I} = \{I_x\}_{x \in V}$ be an interval model for $G$. We show that a consecutive clique arrangement for $G$, that corresponds to the given interval model, can be obtained inductively. If $G$ is complete, then $\langle V \rangle$ is a consecutive clique arrangement for $G$. If $G$ is not complete, let $z$ be a vertex such that the right endpoint of $I_z$ is smallest possible. Then, $N_G[z]$ is a clique of $G$. Let $\langle A_1, \ldots, A_r \rangle$ be a consecutive clique arrangement for $G-z$ that corresponds to $\mathfrak{I} \setminus \{I_z\}$. Then, $A_1$ contains all neighbours of $z$. If $N_G(z) = A_1$, then $\langle N_G[z], A_2, \ldots, A_r \rangle$ is a consecutive clique arrangement for $G$, and if there is $x \in A_1$ that is not a neighbour of $z$, $\langle N_G[z], A_1, \ldots, A_r \rangle$ is a consecutive clique arrangement for $G$ that corresponds to $\mathfrak{I}$.

■

**Corollary 10.7 (Kloks, Kratsch and Spinrad, [52])**
*Let $G = (V, E)$ be an interval graph, and let $\langle A_1, \ldots, A_r \rangle$ be a consecutive clique arrangement for $G$. Then, $S \subseteq V$ is a minimal separator of $G$ if and only if there is $i \in \{1, \ldots, r-1\}$ such that $S = A_i \cap A_{i+1}$.*

**Proof:** Let $i \in \{1, \ldots, r-1\}$. Consider $S =_{\text{def}} A_i \cap A_{i+1}$. Since $A_i$ and $A_{i+1}$ are maximal cliques of $G$, there are vertices $u, v$ of $G$ such that $u \in A_i \setminus A_{i+1}$ and

$v \in A_{i+1} \setminus A_i$. Note that $u$ and $v$ belong to different $S$-full components of $G$. Hence, $S$ is a minimal separator of $G$ due to Lemma 8.1.

For the converse, let $S$ be a minimal separator of $G$. Then, there are vertices $a, b$ of $G$ such that $S$ is a minimal $a, b$-separator of $G$. Let $i, j \in \{1, \ldots, r\}$ such that $a \in A_i$ and $b \in A_j$. Without loss of generality, we assume $i \leq j$, and since $a$ and $b$ are non-adjacent in $G$, $i < j$. Let $C_a$ and $C_b$ be the connected components of $G \setminus S$ containing $a$ and $b$, respectively. Let $i'$ be largest possible such that $A_{i'}$ contains a vertex of $C_a$; similarly, let $j'$ be smallest possible such that $A_{j'}$ contains a vertex of $C_b$. Since $a$ and $b$ belong to different connected components of $G \setminus S$, $i' < j'$. We claim that $S = A_{i'} \cap A_{i'+1}$. Since every vertex of $S$ has neighbours in $C_a$ and $C_b$, it is clear that $S \subseteq A_{i'} \cap A_{i'+1}$. And if there is $x \in (A_{i'} \cap A_{i'+1}) \setminus S$, then $x$ belongs to $C_a$, which contradicts the choice of $i'$.

■

## 10.2 Characterising minimal separators of permutation graphs

Minimal triangulations and minimal separators are connected by an interesting relationship, and this relationship is completely characterised by Theorem 8.15. However, it cannot be expected in general that Theorem 8.15 implies efficient algorithms, since finding all minimal separators, determining their relationships with respect to the crossing relation and generating maximal sets of pairwise non-crossing minimal separators is time-consuming. Nevertheless, minimal separators are of great importance in this chapter. It is interesting that, in case of permutation graphs, they have efficient representations, and they can be found easily in the permutation diagram. For this purpose, Bodlaender, Kloks, Kratsch introduced *scanlines* [8].

**Definition 41** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. A* ***scanline*** *of $G$ is a pair $(a, e)$ where $a, e \in \{0.5, 1.5, \ldots, n+\frac{1}{2}\}$.*

Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$, and let $\mathfrak{D} = \mathfrak{D}(\pi)$ be its permutation diagram. Let $s = (a, e)$ and $s' = (a', e')$ be scanlines of $G$. We say that $s \leq s'$ if and only if $a \leq a'$ and $e \leq e'$; $s < s'$ if and only if $s \leq s'$ and $s \neq s'$. By $\text{int}(s)$, we mean the set of vertices $x \in \{1, \ldots, n\}$ such that $(a-x)(e-\pi^{-1}(x)) < 0$, i.e., $\text{int}(s)$ is the set of vertices scanline $s$ intersects with. In terms of permutation diagrams, $s$ can be thought of as a line segment and $\text{int}(s)$ is the set of vertices intersecting with $s$. We say that a vertex $x$ is *to the left of* $s$ in $\mathfrak{D}$ if it does not intersect with $s$ and is smaller than $a$. Similarly, a vertex $x$ is *to the right of* $s$ in $\mathfrak{D}$ if it does not intersect with $s$ and is greater than $a$. It is easy to see that $\text{int}(s)$ is a separator of $G$, if there are vertices to the left and to the right of $s$. For $C_1, C_2 \subseteq \{1, \ldots, n\}$, $s$ is *between* $G[C_1]$ and $G[C_2]$, if the vertices in $C_1$ are to the left of $s$ and the vertices in $C_2$ are to the right of $s$, or if the vertices in $C_2$ are to the left of $s$ and the vertices in $C_1$ are to the right of $s$. In these cases, $\text{int}(s)$ is an $a, b$-separator of $G$ for every pair $a, b$ of vertices where $a \in C_1$ and $b \in C_2$.

**Definition 42** *Let $G$ be a permutation graph over $\{1, \ldots, n\}$. A scanline $s$ of $G$ is called* ***special*** *if and only if $S =_{\text{def}} \text{int}(s)$ is a minimal separator of $G$ and $s$ is between two $S$-full components of $G$.*

We want to list all special scanlines of a permutation graph. Our algorithm is based on a characterisation of special scanlines that takes into account the near situation in the permutation diagram. The *enclosure* of a scanline $s = (a, e)$ where $a, e \in \{1.5, \ldots, n-\frac{1}{2}\}$ is the set $\text{en}(s) =_{\text{def}} \{a-\frac{1}{2}, a+\frac{1}{2}, \pi(e-\frac{1}{2}), \pi(e+\frac{1}{2})\}$. So, the enclosure may contain between two and four vertices.

**Lemma 10.8** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. A scanline $s$ of $G$ is special if and only if* $\text{en}(s) \cap \text{int}(s) = \emptyset$.

**Proof:** Let $s = (a+\frac{1}{2}, e+\frac{1}{2})$ be a scanline, $1 \leq a, e < n$, and let $S =_{\text{def}} \text{int}(s)$. If $\text{en}(s) \cap S = \emptyset$, then $a$ and $\pi(e)$ belong to one connected component $C_1$ of $G \setminus S$ and $a+1$ and $\pi(e+1)$ belong to another connected component $C_2$ of $G \setminus S$. Since every vertex in $S$ has neighbours in $C_1$ and $C_2$, $C_1$ and $C_2$ are $S$-full components of $G$ and $s$ is between $C_1$ and $C_2$. Hence, $S$ is a minimal separator of $G$, and $s$ is a special scanline. Now, let $\text{en}(s) \cap S \neq \emptyset$. Then, there is $u \in \text{en}(s) \cap S$, and either $u$ has no neighbour in the connected components of $G \setminus S$ to the left of $s$ or $u$ has no neighbour in the connected components of $G \setminus S$ to the right of $s$. Hence, $s$ is not a special scanline. Note that $S$ may not be a minimal separator or that there is no $S$-full component to the left or right of $s$.

■

**Corollary 10.9** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$, and let $m$ be the number of edges of $G$. Then, $G$ has at most* $\min\{n+m, \binom{n}{2}-m\} \leq \frac{1}{4}n\cdot(n+1)$ *special scanlines, and they can be listed in linear time.*

**Proof:** Every special scanline of $G$ can be identified by a point on the upper line and a point on the lower line of the permutation diagram of $G$. For instance, if $s = (a+\frac{1}{2}, e+\frac{1}{2})$ is a special scanline of $G$, then $s$ can be identified by the vertex sets $\{a, \pi(e)\}$ and $\{a, \pi(e+1)\}$. In the former case, these sets can be of cardinality 1 or 2, and if they contain two vertices these vertices are adjacent. In the latter case, the sets always contain exactly two non-adjacent vertices. Hence, $G$ has at most $n + m$ special scanlines and at most $\binom{n}{2} - m$ special scanlines. For the upper bound, note that, if $n \geq 4$, $\min\{n + m, \binom{n}{2} - m\}$ is largest possible if $n + m$ equals $\binom{n}{2} - m$, and this is true for $m = \frac{1}{4}n \cdot (n - 3)$. For listing the special scanlines, simply check, for every vertex $x$ and every edge $uv$ of $G$ where $u < v$, whether $(x+\frac{1}{2}, \pi^{-1}(x)+\frac{1}{2})$ or $(v+\frac{1}{2}, \pi^{-1}(u)+\frac{1}{2})$ are special scanlines, which results in a linear-time listing algorithm.

■

By definition, special scanlines represent minimal separators of permutation graphs. We will show that the converse is equally true, i.e., every minimal separator is represented by a special scanline. This is based on a lemma by Bodlaender, Kloks, Kratsch, that we extend. The construction of the proof is related to the construction of the proof of Corollary 10.7.

**Lemma 10.10 (Bodlaender, Kloks and Kratsch, [8])**
*Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Let $C_1, C_2 \subseteq V$ induce connected subgraphs of $G$. If $N_G[C_1] \cap C_2 = \emptyset$, then there is a special scanline $s$ between $G[C_1]$ and $G[C_2]$ such that $S =_{\text{def}} \text{int}(s)$ is a minimal $u, v$-separator of $G$ for some vertices $u \in C_1$ and $v \in C_2$.*

**Proof:** Since $C_1$ and $C_2$ induce connected subgraphs of $G$ that do not share a vertex, neither do they contain neighbours of each other, we assume $\max C_1 < \min C_2$. Let $u \in C_1$ and $v \in C_2$ be vertices of $G$. Since $S' =_{\text{def}} V \setminus (C_1 \cup C_2)$ is an $x, z$-separator of $G$ for any pair of vertices from $C_1$ and $C_2$, there is a minimal $u, v$-separator $S \subseteq S'$ of $G$. Let $D_1$ and $D_2$ be the connected components of $G \setminus S$ containing the vertices of $C_1$ and $C_2$, respectively. It holds that $D_1$ and $D_2$ are $S$-full components of $G$, and every vertex of $S$ has neighbours in $D_1$ and $D_2$. Let $a =_{\text{def}} \max D_1$ and $e =_{\text{def}} \max \pi^{-1}(D_1)$, and let $s =_{\text{def}} (a+\frac{1}{2}, e+\frac{1}{2})$. Observe that $a < \min D_2$ and $e < \min \pi^{-1}(D_2)$. Then, $S = \text{int}(s)$, and $s$ is a scanline between two $S$-full components of $G$. Hence, $s$ is a special scanline.

■

**Corollary 10.11** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$, and let $S \subseteq \{1, \ldots, n\}$. If $S$ is a minimal separator of $G$, then there is a special scanline $s$ of $G$ such that $S = \text{int}(s)$.*

**Proof:** Let $S$ be a minimal separator of $G$, and let $C_1$ and $C_3$ induce (different) $S$-full components of $G$. Consider $G \setminus S$. Every vertex of $G \setminus S$ between $C_1$ and $C_3$ belongs to an $S$-full component of $G$. Let $C_2$ be the $S$-full component of $G$ containing $\max C_1 + 1$, which does not belong to $S$. Applying Lemma 10.10 to $C_1$ and $C_2$ yields a special scanline $s$ of $G$ such that $S = \text{int}(s)$.

■

In case of permutation graphs, crossing of minimal separators, which is an important property in connection with minimal triangulations (Theorem 8.15), can be determined in the permutation diagram and translates into a property of special scanlines. So, special scanlines are "full" representations of minimal separators of permutation graphs, since they preserve all information, that we need, of the corresponding separators.

**Definition 43** *Two scanlines $s_1$ and $s_2$ of a permutation graph $G = G(\pi)$* ***intersect*** *if and only if neither $s_1 \leq s_2$ nor $s_2 < s_1$.*

In other words, intersection of scanlines, that do not share common endpoints, has the original geometric meaning.

**Lemma 10.12** *Let $G = G(\pi)$ be a permutation graph over $\{1, \dots, n\}$. Let $s_1$ and $s_2$ be special scanlines of $G$, and let $S_1 =_{\text{def}} \text{int}(s_1)$ and $S_2 =_{\text{def}} \text{int}(s_2)$.*

*(1) If $s_1$ and $s_2$ do not intersect then $S_1$ and $S_2$ are non-crossing.*

*(2) If $s_1$ and $s_2$ do intersect then $S_1$ and $S_2$ cross .*

**Proof:** By definition, $S_1$ and $S_2$ are minimal separators of $G$. Let $s_1$ and $s_2$ do not intersect. We assume $s_1 < s_2$. Let $C_1, \dots, C_\ell$ be the connected components of $G \setminus S_1$, $\ell \geq 2$, where the vertices in $C_i$ are smaller than the vertices in $C_{i+1}$ for all $i \in \{1, \dots, \ell{-}1\}$. Suppose $S_1$ and $S_2$ cross, i.e., there are two connected components of $G \setminus S_1$ that contain vertices from $S_2$. So, there is $r \geq 1$ such that $s_2$ intersects with vertices from $C_r$ and $C_{r+1}$. Since $s_1 < s_2$, the vertices in $\text{en}(s_2)$ that are to the right of $s_2$ are not contained in $\text{int}(s_1)$. Hence, there is a path between a vertex from $C_r$ and a vertex from $C_{r+1}$, and $C_r$ and $C_{r+1}$ cannot be in different connected components of $G \setminus S_1$, which contradicts the assumption that $S_1$ and $S_2$ cross. For the converse, let $s_1$ and $s_2$ intersect. Observe that the vertices of $\text{en}(s_1)$ belong to two connected components of $G \setminus S_1$, and one vertex from each of these components belongs to $S_2$. Then, $S_1$ and $S_2$ cross.

■

We remark that the listing algorithm of Corollary 10.9 does not output every minimal separator (in form of a special scanline) exactly once, since one minimal separator may be represented by different scanlines. In [64], a linear-time algorithm is given that selects exactly one special scanline for each minimal separator, so that the number of minimal separators of a permutation graph can be computed in linear time. For general graphs, listing the minimal separators is an interesting problem. Kloks and Kratsch showed that this can be done in polynomial time for a graph class, if the numbers of minimal separators of its graphs are polynomially bounded in the numbers of vertices [48].

## 10.3 Potential maximal cliques of permutation graphs

Instead of using the characterisation of minimal triangulations by maximal sets of pairwise non-crossing minimal separators, Bouchitté and Todinca introduced *potential maximal cliques* to solve problems like treewidth and minimum fill-in [13]. Potential maximal cliques are sets of vertices, and they turn out very useful also for our problems. In this section, we will characterise potential maximal cliques of permutation graphs.

**Definition 44** *Let $G = (V, E)$ be a graph. A set $C \subseteq V$ of vertices of $G$ is a* ***potential maximal clique*** *of $G$ if and only if there is a minimal triangulation $H$ of $G$ such that $C$ is a maximal clique of $H$.*

For our purposes, we need two further scanlines that we declare to be special. Let $G = G(\pi)$ be a permutation graph over $\{1, \dots, n\}$. Then, the scanlines $\mathrm{s}_0 =_{\mathrm{def}} (\frac{1}{2}, \frac{1}{2})$ and $\mathrm{s}_e^n =_{\mathrm{def}} (n+\frac{1}{2}, n+\frac{1}{2})$ are also special. Let $s_1$ and $s_2$ be two special scanlines of $G$ such that $s_1 < s_2$. By $G[s_1, s_2]$ we denote the subgraph of $G$ induced by the vertices that intersect with $s_1$ or $s_2$ or that lie between $s_1$ and $s_2$, i.e., to the right of $s_1$ and to the left of $s_2$. With $G[s_1, s_2]$ we associate the permutation diagram of $G$ reduced to only the vertices of $G[s_1, s_2]$. The non-intersecting scanlines $s_1$ and $s_2$ are *neighbours* if there is no special scanline $s'$ of $G$ such that $s_1 < s' < s_2$. We say that $s_1$ is a *left neighbour* of $s_2$ and $s_2$ is a *right neighbour* of $s_1$. If $s_1$ and $s_2$ have a common endpoint they are *close* neighbours. By $N_\pi^<(s)$ we denote the set of left neighbours of special scanline $s$ of $G$.

**Lemma 10.13** *Let $G = G(\pi)$ be a permutation graph over $\{1, \dots, n\}$, and let $m$ be the number of edges of $G$. Then, $G$ has at most $2m$ pairs of special scanlines that are close neighbours.*

**Proof:** Let $s = (a, e)$ be a special scanline of $G$. There is at most one special scanline $s' = (a, e')$ of $G$ such that $s$ and $s'$ are neighbours and $e' < e$. Similarly, there is at most one special scanline $s'' = (a'', e)$ of $G$ such that $s$ and $s''$ are neighbours and $a'' < a$. Note that in both cases the enclosure of $s$ must contain two vertices that are to the right of $s$, and these vertices are adjacent.

■

**Lemma 10.14** *Let $G = G(\pi) = (V, E)$ be a permutation graph over $\{1, \dots, n\}$. Let $s_1$, $s_2$, $t_1$ and $t_2$ be special scanlines of $G$ where $s_1 \in N_\pi^<(s_2)$ and $t_1 \in N_\pi^<(t_2)$. Let $S_1 =_{\mathrm{def}} \mathrm{int}(s_1)$ and $S_2 =_{\mathrm{def}} \mathrm{int}(s_2)$, and let $C =_{\mathrm{def}} V(G[s_1, s_2])$. Then:*

*(1)* $u \in C \setminus (S_1 \cup S_2) \implies N_G[u] = C$

*(2)* $u \in S_1 \setminus S_2$ and $v \in S_2 \setminus S_1 \implies uv \in E$

*(3)* $C = V(G[t_1, t_2]) \implies t_1 = s_1$ and $t_2 = s_2$ .

**Proof:** Let $u, v \in C$ such that $uv \notin E$. Let $u \in C \setminus (S_1 \cup S_2)$. If $v \in C \setminus (S_1 \cup S_2)$, there is a special scanline between $G[\{u\}]$ and $G[\{v\}]$ by Lemma 10.10 that does not intersect with neither $s_1$ nor $s_2$. If $v \in S_1$, there is a vertex $w$ in the enclosure of $s_1$ that is to the left of $s_1$ and adjacent with $v$ such that there is a special scanline between $G[\{v, w\}]$ and $G[\{u\}]$ by Lemma 10.10, and this scanline does not intersect with neither $s_1$ nor $s_2$. The case $v \in S_2$ is similar to $v \in S_1$. Since $s_1$ and $s_2$ are neighbours there cannot be a special scanline between them, and statement (1) holds. If $u \in S_1 \setminus S_2$ and $v \in S_2 \setminus S_1$, there are vertices $x \in \mathrm{en}(s_1) \setminus C$ and $z \in \mathrm{en}(s_2) \setminus C$ such that $ux \in E$ and $vz \in E$ and there is a special scanline $s$ between $s_1$ and $s_2$ due to Lemma 10.10, applied to $G[\{u,x\}]$ and $G[\{v,z\}]$. Note that Lemma 10.10 can be applied, since $\{v,z\} \cap N_G[\{u,x\}] = \emptyset$. However, $s$ cannot exist, since $s_1$ and $s_2$ are neighbours, and statement (2) holds. Note that this case is not possible, if $s_1 = \mathrm{s}_0$ or $s_2 = \mathrm{s}_e^n$, since $S_1 = \emptyset$ or $S_2 = \emptyset$.

For statement (3) observe that $t_1$ does not intersect with neither $s_1$ nor $s_2$, since otherwise $C' =_{\text{def}} V(G[t_1,t_2])$ would contain vertices from the enclosure of $s_1$ or $s_2$. Analogously, $t_2$ does not intersect with neither $s_1$ nor $s_2$. So, $\{s_1,s_2,t_1,t_2\}$ is a set of pairwise non-intersecting special scanlines. Since $C'$ must contain a vertex to the right of $s_1$ that belongs to the enclosure of $s_1$, $s_1 < t_2$; similarly, $t_1 < s_2$, and this is only possible if $t_1 = s_1$ and $t_2 = s_2$.

■

Let $s_1$ and $s_2$ be special scanlines of $G = G(\pi)$ where $s_1 \in N_\pi^{<}(s_2)$. A vertex $x \in \{1,\ldots,n\}$ is an *inner vertex* of $G[s_1,s_2]$ if $x \notin \text{int}(s_1) \cup \text{int}(s_2)$ and $x \in V(G[s_1,s_2])$. Inner vertices play a special role in our analysis, since $N_G[x] = V(G[s_1,s_2])$ by Lemma 10.14. For a potential maximal clique $C$ of $G$, we say that $x \in C$ is an *inner vertex* of $C$ if $N_G[x] = C$. We prove a useful characterisation of potential maximal cliques with inner vertices.

**Lemma 10.15** *Let $G = (V,E)$ be a graph, and let $x$ be a vertex of $G$. $N_G[x]$ is a potential maximal clique of $G$ if and only if there are no vertices $a,b \in N_G(x)$ such that $N_G[x] \setminus \{a,b\}$ is an $a,b$-separator of $G$.*

**Proof:** Let $C =_{\text{def}} N_G[x]$ be a potential maximal clique of $G$, and let $H$ be a minimal triangulation of $G$ containing $C$ as maximal clique. Let $a$ and $b$ be vertices in $C$ that are non-adjacent in $G$. Since $C$ is a clique in $H$, $ab$ is an edge in $H$, and there are vertices $u,v$ such that $ab$ is unique chord in cycle $(a,u,b,v)$ in $H$ due to Theorem 8.14. Without loss of generality, $u$ does not belong to $C$ and $v = x$. Hence, every $u,x$-separator of $H$ must contain $a$ and $b$. Note that $N_G(x)$ is a $u,x$-separator of $H$. Then, there is a minimal $u,x$-separator $T \subseteq N_G(x)$ of $H$. Due to Theorem 8.16, $T$ is a minimal $u,x$-separator of $G$, and $T$ contains both $a$ and $b$. This means that there are $a,u$- and $b,u$-paths in $G$ that do not contain vertices from $N_G[x]$ except for $a$ and $b$, respectively. Then, $N_G[x] \setminus \{a,b\}$ is not an $a,b$-separator of $G$.

For the converse, let $N_G(x)$ contain no vertices $a,b$ such that $N_G[x] \setminus \{a,b\}$ is an $a,b$-separator of $G$. In other words, for every pair $a,b$ of non-adjacent vertices in $N_G(x)$, there is an $a,b$-path in $G$ containing no vertex from $N_G[x] \setminus \{a,b\}$. Consider triangulation $H' =_{\text{def}} G \cup F'$ of $G$ where $F' =_{\text{def}} \{uv \notin E : u,v \neq x\}$, i.e., $H' - x$ is complete and $N_{H'}(x) = N_G(x)$. Let $H \subseteq H'$ be a minimal triangulation of $G$. We prove that $N_G[x]$ is a clique in $H$. Suppose there are vertices $a,b \in N_G(x)$ that are non-adjacent in $H$. By assumption, there is an $a,b$-path $P = (a,u_1,\ldots,u_r,b)$ in $H$ that does not contain vertices from $N_G[x]$ except for $a$ and $b$. Without loss of generality, $P$ is shortest possible. It holds that $r \geq 1$. Then, $(x,a,u_1,\ldots,u_r,b)$ is a chordless cycle of length at least 4, which contradicts $H$ being chordal. Hence, $a$ and $b$ are adjacent in $H$, and $N_G[x]$ is a clique in $H$. Since $N_H[x] = N_G[x]$, $N_G[x]$ is a maximal clique in $H$, therefore a maximal clique in a minimal triangulation of $G$, i.e., a potential maximal clique of $G$.

■

Starting from minimal triangulations, potential maximal cliques with inner vertices are easily identifiable.

**Lemma 10.16** *Let $G = (V, E)$ be a graph, and let $H$ be a minimal triangulation of $G$. Let $u \in V$ such that there is only one maximal clique $C$ in $H$ containing $u$. Then, $u$ is an inner vertex of $C$.*

**Proof:** Since $\{u, v\}$ for every $v \in N_G(u)$ is a clique in $H$, $N_G[u] \subseteq C$. Suppose there is $v \in C$ such that $uv \notin E$. Since $uv \in E(H)$, $uv$ is unique chord in a cylce $(u, x, v, z)$ in $H$ by Theorem 8.14. But since $x, z \in C$, $x$ and $z$ are adjacent in $H$, hence $(u, x, v, z)$ contains two chords in $H$, which contradicts the assumption.

■

**Lemma 10.17** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Then, $G$ has at most $n$ pairs of special scanlines that are neighbours but not close neighbours.*

**Proof:** Let $s_1$ and $s_2$ be special scanlines of $G$ that are neighbours but not close neighbours. Then, $G[s_1, s_2]$ must contain an inner vertex $x$; otherwise, i.e., every vertex of $G[s_1, s_2]$ intersects with $s_1$ or $s_2$, $G$ would have a special scanline that is close to $s_1$ and $s_2$, hence between $s_1$ and $s_2$ (this also follows from Lemma 10.14, statement (2)). Let $(t_1, t_2)$ be a pair of special scanlines of $G$ where $t_1 < t_2$ and $t_1$ and $t_2$ are neighbours but not close neighbours such that $x$ is inner vertex of $G[t_1, t_2]$. Suppose $s_1$ and $t_1$ intersect. There are two non-adjacent vertices of the enclosure of $t_1$ that belong to $\text{int}(s_1)$, and due to Lemma 10.14, they are neighbours of $x$. Hence, $t_1$ intersects with $x$, which is not possible and $s_1 = t_1$. Similarly, $s_2$ and $t_2$ do not intersect. Hence, $\{s_1, s_2, t_1, t_2\}$ is a set of pairwise non-intersecting scanlines of $G$, since $x$ "separates" $s_1$ and $t_1$ from $s_2$ and $t_2$. Since $s_1$ and $s_2$ as well as $t_1$ and $t_2$ are neighbours, it follows that $s_1 = t_1$ and $s_2 = t_2$. Thus, for every vertex $u$ of $G$, there is at most one pair $(s_1, s_2)$ of special scanlines of $G$ that are neighbours but not close neighbours such that $u$ is inner vertex of $G[s_1, s_2]$.

■

The main result of this section is a characterisation of potential maximal cliques of permutation graphs in terms of special scanlines. We will consider two cases: potential maximal cliques with and without inner vertices.

**Lemma 10.18** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Let $C \subseteq \{1, \ldots, n\}$ be a potential maximal clique of $G$ that has an inner vertex. Then, there are special scanlines $s_1, s_2$ of $G$ such that $s_1 \in N_\pi^{\leq}(s_2)$ and $C = V(G[s_1, s_2])$, where $s_1$ and $s_2$ are not close neighbours.*

**Proof:** By definition, there is a vertex $x$ of $G$ such that $N_G[x] = C$. We first define two scanlines of $G$ as follows. Let

$$\begin{aligned}
a_1 &=_{\text{def}} \max\{a < x : a \notin N_G(x)\} \cup \{0\} \\
a_2 &=_{\text{def}} \min\{a > x : a \notin N_G(x)\} \cup \{n{+}1\} \\
e_1 &=_{\text{def}} \max\{i < \pi^{-1}(x) : \pi(i) \notin N_G(x)\} \cup \{0\} \\
e_2 &=_{\text{def}} \min\{i > \pi^{-1}(x) : \pi(i) \notin N_G(x)\} \cup \{n{+}1\}\,,
\end{aligned}$$

and let $s_1 =_{\text{def}} (a_1{+}\frac{1}{2}, e_1{+}\frac{1}{2})$ and $s_2 =_{\text{def}} (a_2{-}\frac{1}{2}, e_2{-}\frac{1}{2})$. By construction, $s_1$ and $s_2$ are special scanlines of $G$ ($a_1{+}1, \ldots, a_2{-}1$ and $\pi(e_1+1), \ldots, \pi(e_2-1)$ are contained in $N_G[x]$). Furthermore, $s_1 < s_2$. Suppose there is a special scanline $s$ of $G$ such that $s_1 < s < s_2$. By definition of $a_1, a_2, e_1, e_2$, $s$ must be close neighbour of $s_1$ and $s_2$. Then, $\text{en}(s)$ contains exactly two vertices from $N_G(x)$, and they are non-adjacent. Let $\{a, b\} = \text{en}(s) \cap N_G(x)$. By construction of $s_1$ and $s_2$, $N_G[x] \setminus \{a, b\}$ is an $a, b$-separator of $G$. Hence, $N_G[x]$ cannot be a potential maximal clique of $G$ due to Lemma 10.15, which contradicts the assumption. So, $s_1$ and $s_2$ are neighbouring special scanlines without common endpoint and $C = G[s_1, s_2]$.

■

For the case of potential maximal cliques without inner vertices, we have to know more about the structure of minimal triangulations of permutation graphs. However, minimal triangulations of permutation graphs are considered not before the following section. To preserve the composition of this thesis, we use a result here that is stated and proved later: Minimal triangulations of permutation graphs are interval graphs (Corollary 10.24).

**Lemma 10.19** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Let $C \subseteq \{1, \ldots, n\}$ be a potential maximal clique of $G$ without inner vertices. Then, there are minimal separators $S_1, S_2$ of $G$ such that $C = S_1 \cup S_2$. Furthermore, there are special scanlines $s_1, s_2$ of $G$ such that $s_1 \in N_G^{<}(s_2)$, $\{S_1, S_2\} = \{\text{int}(s_1), \text{int}(s_2)\}$ and $C = V(G[s_1, s_2])$. In particular, $s_1$ and $s_2$ are close neighbours.*

**Proof:** Let $H$ be a minimal triangulation of $G$ such that $C$ is a maximal clique of $H$. By Corollary 10.24, $H$ is an interval graph. Let $\langle A_1, \ldots, A_k\rangle$, $k \geq 1$, be a consecutive clique arrangement for $H$. Let $i \in \{1, \ldots, k\}$ such that $A_i = C$. Since $A_1$ and $A_k$ have inner vertices, $k \geq 3$ and $1 < i < k$. Let $S_1 =_{\text{def}} A_{i-1} \cap A_i$ and $S_2 =_{\text{def}} A_i \cap A_{i+1}$. By Corollary 10.7, $S_1$ and $S_2$ are minimal separators of $H$, and by Theorem 8.16, $S_1$ and $S_2$ are minimal separators of $G$. Since there is no vertex $x$ in $C$ such that $N_G[x] = C$, every vertex of $C$ is contained in $A_{i-1} \cup A_{i+1}$. Hence, $C = S_1 \cup S_2$. Due to maximality of $C$, $S_1 \not\subseteq S_2$ and $S_2 \not\subseteq S_1$. Let $u \in C \setminus S_1$, $v \in A_{i-1} \setminus S_1$ and $w \in C \setminus S_2$, $x \in A_{i+1} \setminus S_2$ (see Figure 21). We assume $v < u$; if $u < v$ we use $\langle A_k, \ldots, A_1\rangle$ as consecutive clique arrangement for $H$ and the same vertices with their new meanings. It holds that $v, w, u, x$ induce a chordless path in $H$, and $u$ and $w$ are adjacent in $G$. We want to show that $w < x$. Suppose $w > x$. Remember that $S_1$ and $S_2$ are minimal $u, v$- and $w, x$-separators in $H$, respectively, and in $G$ by Theorem 8.16. If $x < v$, i.e., $x < v < u$ and $u$ and $x$ are non-adjacent in $G$, then the $u, v$-separator $S_1$ also separates $u$ and $x$, which contradicts Theorem 8.16, since $u$ and $x$ are contained in the same connected component of $H \setminus S_1$. If $v < x$, i.e., $v < x < w$ and $w$ and $v$ are non-adjacent in $G$, similarly, $S_2$ also separates $w$ and $v$ contradicting Theorem 8.16. Hence, $w < x$. Due to Theorem 8.16, $C \setminus S_1$ is contained in a connected component of $G \setminus S_1$ induced by $D_1$.

[We construct two scanlines $s_1$ and $s_2$ of $G$, that shall define the potential maximal clique $C$.]
Let $a_1 =_{\text{def}} \min(D_1)$ and $e_1 =_{\text{def}} \min \pi^{-1}(D_1)$ and $s_1 =_{\text{def}} (a_1{-}\frac{1}{2}, e_1{-}\frac{1}{2})$. Observe that $a_1{-}1$ and $\pi(e_1{-}1)$ exist and $v \leq a_1{-}1$ and $\pi^{-1}(v) \leq e_1{-}1$ and $a_1{-}1$ and $\pi(e_1{-}1)$ are to the left of $s_1$. Since $a_1 \in D_1$ and $\pi(e_1) \in D_1$, $a_1$ and $\pi(e_1)$ are to the right of $s_1$, hence $s_1$ is a special scanline of $G$ due to Lemma 10.8. Let $D'_1$ induce the connected component of $G \setminus S_1$ containing $v$. Since $S_1$ is a minimal $u, v$-separator in $H$ and $G$, every vertex in $S_1$ has a neighbour in $D_1$ and $D'_1$, and $S_1 = \text{int}(s_1)$. By a similar construction using $S_2$, define $D_2$ and $D'_2$ and $s_2 =_{\text{def}} (a_2{+}\frac{1}{2}, e_2{+}\frac{1}{2})$, where $a_2 =_{\text{def}} \max(D_2)$ and $e_2 =_{\text{def}} \max \pi^{-1}(D_2)$. It holds that $S_2 = \text{int}(s_2)$. Since $u < w$ or $\pi^{-1}(u) < \pi^{-1}(w)$, $a_1 \leq u < w \leq a_2$ or $e_1 \leq \pi^{-1}(u) < \pi^{-1}(w) \leq e_2$, and since $S_1$ and $S_2$ are non-crossing minimal separators of $G$ due to Theorem 8.15, $s_1 < s_2$ by Lemma 10.12. Hence, $C \subseteq V(G[s_1, s_2])$.

[Scanlines $s_1$ and $s_2$ indeed define $C$.]
Since $H$ is a minimal triangulation of $G$, for every pair $a, b$ of vertices of $G$ where $a \in S_1 \setminus S_2$ and $b \in S_2 \setminus S_1$, $a$ and $b$ are adjacent in $G$. Imagine the situation in the permutation diagram of $G$. Suppose there is $z \in V(G[s_1, s_2]) \setminus C$. Then, $z$ must be adjacent with all vertices in $S_1$ or all vertices in $S_2$ in $G$. Since $z \notin C$, $z \in A_1 \cup \cdots \cup A_{i-1}$ or $z \in A_{i+1} \cup \cdots \cup A_k$. In the former case, $S_1$ is a $z, u$-separator, in the latter case, $S_2$ is a $w, z$-separator in $H$ and therefore in $G$. Hence, $z$ has no neighbours in $S_2 \setminus S_1$, which means that $z$ is not contained in $D_1$ and to the left of $s_1$, or in $S_1 \setminus S_2$, which means that $z$ is not contained in $D_2$ and to the right of $s_2$. So, $z \notin G[s_1, s_2]$ and $C = V(G[s_1, s_2])$. Furthermore, $s_1$ and $s_2$ have a common endpoind.

[Scanlines $s_1$ and $s_2$ are neighbours.]
If $a_1 = a_2 + 1$, then there is $j \in \{e_1, \ldots, e_2{-}1\}$ such that $\pi(e_1), \ldots, \pi(j) \in S_2$ and $\pi(j{+}1), \ldots, \pi(e_2) \in S_1$. If $e_1 = e_2 + 1$, then there is $j \in \{a_1, \ldots, a_2{-}1\}$ such that $a_1, \ldots, j \in S_2$ and $j{+}1, \ldots, a_2 \in S_1$. Then, there cannot be a special scanline $s$ such that $s_1 < s < s_2$, and $s_1 \in N^{<}_{\pi}(s_2)$.

■

Since scanlines $s_1$ and $s_2$ of Lemma 10.19 are unique, $S_1$ and $S_2$ of Lemma 10.19 are also unique up to indices according to Lemma 10.19.

**Theorem 10.20** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. A vertex set $C \subseteq \{1, \ldots, n\}$ is a potential maximal clique of $G$ if and only if there are special scanlines $s_1$ and $s_2$ of $G$ such that $s_1 \in N^{<}_{\pi}(s_2)$ and $C = V(G[s_1, s_2])$.*

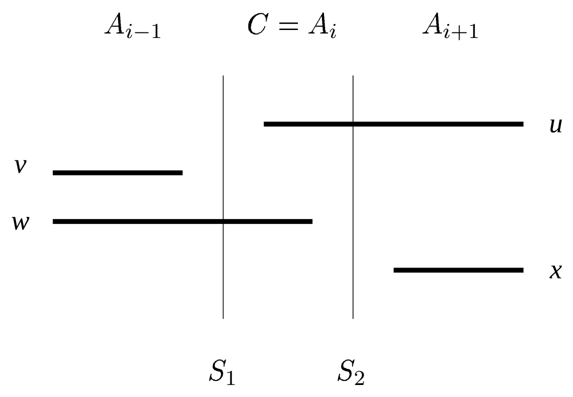


**Figure 21** The choice of vertices $u, v, w, x$ in the proof of Lemma 10.19. Intervals crossing the thin lines represent vertices contained in separators $S_1$ or $S_2$.

**Proof:** The "only if" part of the theorem holds by Lemmata 10.18 and 10.19. For the "if" part of the theorem, let $s_1$ and $s_2$ be special scanlines such that $s_1 \in N_\pi^<(s_2)$, and let $C =_{\text{def}} V(G[s_1, s_2])$. Observe that there are $u, v \in C$ such that $u \notin \text{int}(s_1)$ and $v \notin \text{int}(s_2)$; $u$ and $v$ may be identical. Let $G'$ emerge from $G$ by completing into cliques $\text{int}(s_1)$ and $\text{int}(s_2)$. Due to Lemma 10.14, $C$ induces a clique in $G'$. Since no vertex to the left of $s_1$ is adjacent with $u$ and no vertex to the right of $s_2$ is adjacent with $v$, $C$ is a maximal clique of $G'$. Obtain $G''$ from $G'$ by making adjacent all vertices to the left of $s_1$ with all vertices in $\text{int}(s_1)$ and all vertices to the right of $s_2$ with all vertices in $\text{int}(s_2)$. Still, $C$ is a maximal clique of $G''$. Now, obtain $H'$ from $G''$ by making complete every connected component of $G'' \setminus C$. Note that $H'$ is a triangulation of $G''$, and $C$ is a maximal clique of $H'$. $H'$ is also a triangulation of $G'$, and there is a subgraph $H$ of $H'$ that is a minimal triangulation of $G'$. $C$ is a maximal clique of $H$, and since every minimal triangulation of $G'$ is a minimal triangulation of $G$, $C$ is a maximal clique in a minimal triangulation of $G$, and $C$ is a potential maximal clique of $G$.

■

We deduce an interesting upper bound for the number of potential maximal cliques of permutation graphs.

**Corollary 10.21** *A permutation graph on $n \geq 1$ vertices and with $m$ edges has at most $n + 2m$ potential maximal cliques.*

**Proof:** Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Let $G[s_1, s_2]$ be a potential maximal clique of $G$, $s_1 < s_2$ and $s_1 \in N_\pi^<(s_2)$. Then, either $s_1$ and $s_2$ are close neighbours or $G[s_1, s_2]$ has an inner vertex. Applying Lemmata 10.13 and 10.17, we obtain the desired result.

■

## 10.4 Characterisations of minimal triangulations of AT-free graphs

The corresponding sections for $2K_2$-free graphs and AT-free claw-free graphs each contain two characterisations of minimal triangulations: one characterises the whole set of minimal triangulations and the other characterises the set of minimal triangulations of a single graph. This section differs from that structure: In the first part, we will characterise the minimal triangulations of AT-free graphs, and only in the second part, we will concentrate on permutation graphs. At least in this second part, we will meet the general setting.

The characterisation of minimal triangulations of AT-free graphs can be understood as the end of a long series of results about minimal triangulations of AT-free graphs. Bodlaender and Möhring showed that minimal triangulations of cographs are interval graphs [10]. Then, Bodlaender, Kloks, Kratsch showed that minimal triangulations of permutation graphs are interval graphs [8]. Habib and Möhring continued with the result for cocomparability graphs [38]. And finally, Möhring proved that minimal triangulations of arbitrary AT-free graphs are interval graphs [65]. Parra and Scheffler completed this result to a characterisation [69]. An alternative proof of Möhring's theorem was given by Kloks, Kratsch, Spinrad [52], on which the proof below is based.

**Theorem 10.22 (Möhring, [65]; Parra and Scheffler, [69])**
*A graph is AT-free if and only if all its minimal triangulations are AT-free.*

**Proof:** Let $G = (V, E)$ be a graph that contains an asteroidal triple formed by the three vertices $u, v, w$. Let $H'$ emerge from $G$ by completing into a clique $V \setminus \{u, v, w\}$. Since $u, v, w$ are pairwise non-adjacent, $H'$ is chordal, and $u, v, w$ form an asteroidal triple of $H'$. Then, there is a minimal triangulation $H$ of $G$ that is a subgraph of $H'$ and therefore contains an asteroidal triple formed by $u, v, w$. We conclude that every graph containing an asteroidal triple has a minimal triangulation that contains an asteroidal triple.

For the converse, let $G = (V, E)$ be an AT-free graph, and let $H$ be a minimal triangulation of $G$. Suppose $H$ contains an asteroidal triple formed by the vertices $u, v, w$. Since $N_H(u)$, $N_H(v)$ and $N_H(w)$ are $u, v$-, $v, w$- and $w, u$-separators of $H$, respectively, there are minimal $u, v$-, $v, w$- and $w, u$-separators $S_u$, $S_v$ and $S_w$, respectively, such that $S_u \subseteq N_H(u)$, $S_v \subseteq N_H(v)$ and $S_w \subseteq N_H(w)$. By Theorem 8.16, $S_u$, $S_v$ and $S_w$ are minimal separators of $G$, and $\{u\}$ and $\{v, w\}$ belong to different connected components of $G \setminus S_u$. Hence, there is a $v, w$-path in $G$ that does not contain a neighbour of $u$. Similarly for $S_v$ and $v$ as well as $S_w$ and $w$. Then, $u, v, w$ form an asteroidal triple of $G$, which contradicts our assumption, and $H$ is AT-free.

■

**Corollary 10.23** *A graph is AT-free if and only if all its minimal triangulations are interval graphs.*

**Proof:** Due to Theorem 10.4, interval graphs are exactly the chordal AT-free graphs, and the corollary follows from Theorem 10.22.

■

**Corollary 10.24** *Every minimal triangulation of a permutation graph is an interval graph.*

**Proof:** Let $G = G(\pi)$ be a permutation graph. Due to Lemma 10.3, permutation graphs are cocomparability graphs, and due to Lemma 10.1, cocomparability graphs are AT-free. Applying Corollary 10.23, we obtain that minimal triangulations of permutation graphs are interval graphs.

■

Our characterisation of minimal triangulations of a given permutation graph is based on potential maximal cliques. Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. The *potential maximal cliques graph* of $G$ is a directed graph that is denoted by $\mathcal{PC}(\pi)$ and defined as follows: $\mathcal{PC}(\pi)$ has a vertex for every special scanline of $G$ (including $\mathrm{s}_0$ and $\mathrm{s}_e^n$), that is labelled with this scanline, and there is an arc from vertex $u$ to vertex $v$ if and only if $s_u \in N_\pi^{<}(s_v)$, where $s_u$ and $s_v$ denote the special scanlines the vertices $u$ and $v$ are labelled with, respectively. Hence, there is a 1-to-1 correspondence between the potential maximal cliques of $G$ and the arcs in $\mathcal{PC}(\pi)$ due to Theorem 10.20.

**Lemma 10.25** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$.*

*(1) $\mathcal{PC}(\pi)$ is acyclic.*

*(2) Let $k \geq 1$, and let $x_0, \ldots, x_k$ be (distinct) vertices of $\mathcal{PC}(\pi)$. It holds that $(x_0, \ldots, x_k)$ is a maximal path in $\mathcal{PC}(\pi)$ if and only if $\mathrm{s}_0, s_1, \ldots, s_{k-1}, \mathrm{s}_e^n$ are the labels of $x_0, \ldots, x_k$, respectively, and $\{\mathrm{s}_0, s_1, \ldots, s_{k-1}, \mathrm{s}_e^n\}$ is a maximal set of pairwise non-intersecting special scanlines of $G$.*

**Proof:** We prove statement (1). Suppose $(x_1, \ldots, x_r)$, $r \geq 3$ is a cycle of $\mathcal{PC}(\pi)$. Let $s_1, \ldots, s_r$ be the scanlines the vertices of the cycle are labelled with. By definition of $\mathcal{PC}(\pi)$, $s_1 < \cdots < s_r < s_1$, which is not possible, since $<$ defines a partial order on the set of scanlines of $G$. Hence, $\mathcal{PC}(\pi)$ is acyclic.

For statement (2), let $(x_0, \ldots, x_k)$ be a maximal path in $\mathcal{PC}(\pi)$. Obviously, the labels of $x_0$ and $x_k$ are $\mathrm{s}_0$ and $\mathrm{s}_e^n$, respectively, since $\mathrm{s}_0 \leq s \leq \mathrm{s}_e^n$ for every special scanline $s$ of $G$. Let $s_1, \ldots, s_{k-1}$ be the labels of $x_1, \ldots, x_{k-1}$, respectively. By definition of $\mathcal{PC}(\pi)$, $\{\mathrm{s}_0, s_1, \ldots, s_{k-1}, \mathrm{s}_e^n\}$ is a maximal set of pairwise non-intersecting special scanlines of $G$. Conversely, let $S =_{\mathrm{def}} \{s_0, \ldots, s_k\}$ be a maximal set of pairwise non-intersecting special scanlines of $G$ where $s_0 < \cdots < s_k$. Then, $s_{i-1} \in N_\pi^{<}(s_i)$ for all $i \in \{1, \ldots, k\}$, and $s_0 = \mathrm{s}_0$ and $s_k = \mathrm{s}_e^n$. This, however, uniquely defines a maximal path in $\mathcal{PC}(\pi)$.

■

The potential maximal cliques graph is a central structure of our characterisation of minimal triangulations of a permutation graph. We will show that every maximal set of pairwise non-intersecting special scanlines corresponds to a minimal triangulation, and using Lemma 10.25, the correspondence with the maximal paths of the potential maximal cliques graph is obvious.

Let $G = G(\pi) = (V, E)$ be a permutation graph over $\{1, \ldots, n\}$. For every maximal path in $\mathcal{PC}(\pi)$, we define an interval graph. Let $P = (x_0, \ldots, x_k)$ be a maximal path in $\mathcal{PC}(\pi)$, and let $s_0, \ldots, s_k$ be the special scanlines of $G$ the vertices $x_0, \ldots, x_k$ are labelled with, respectively. Due to Lemma 10.25, $\{s_0, \ldots, s_k\}$ is a maximal set of non-intersecting special scanlines of $G$ and $s_{i-1} \in N_\pi^{<}(s_i)$ for all $i \in \{1, \ldots, k\}$. Furthermore, $s_0 = \mathrm{s}_0$ and $s_k = \mathrm{s}_e^n$. Every vertex $x$ of $G$ is assigned an interval $I_x = [\ell_x, r_x]$ in the following way:

$$\begin{aligned}\ell_x &=_{\text{def}} \max\{i : s_i = (a_i, e_i) \in S \text{ and } a_i < x \text{ and } e_i < \pi^{-1}(x)\} + 1\\ r_x &=_{\text{def}} \min\{i : s_i = (a_i, e_i) \in S \text{ and } a_i > x \text{ and } e_i > \pi^{-1}(x)\}.\end{aligned}$$

We can say that $\ell_x$ determines the "rightmost" scanline to the left of $x$, and similarly, $r_x$ determines the "leftmost" scanline to the right of $x$. It is easy to verify that $1 \leq \ell_x \leq r_x \leq k$ always holds. Let $H(P)$ be the interval graph defined by $\{I_x\}_{x \in V}$, and let $A_i =_{\text{def}} \{x \in V : \ell_x \leq i \leq r_x\}$ for $i \in \{1, \dots, k\}$. For being more precise, we should write $H_\pi(P)$ instead of $H(P)$. But the context will always be clear, so that we avoid $\pi$ as further parameter.

**Lemma 10.26** *(1) $A_i = V(G[s_{i-1}, s_i])$ for $i \in \{1, \dots, k\}$.*

*(2) $H(P)$ is a minimal triangulation of $G$.*

**Proof:** For statement (1), let $i \in \{1, \dots, k\}$, and let $s_{i-1} = (a, e)$ and $s_i = (a', e')$. By definition, it holds that $x \in V(G[s_{i-1}, s_i])$ if and only if $x$ is between $s_{i-1}$ and $s_i$ or $x \in \text{int}(s_{i-1}) \cup \text{int}(s_i)$. Remember that both cases cannot happen simultaneously. The former is the case if and only if $(i-1)+1 = \ell_x = r_x = i$, the latter is the case if and only if $\ell_x < (i-1)+1 = i \leq r_x$ or $\ell_x \leq (i-1)+1 = i < r_x$. Hence, $x \in V(G[s_{i-1}, s_i])$ if and only if $x \in A_i$.

For statement (2), let $uv \in E$, $u < v$. It holds that $u \in A_{\ell_v}$ or $v \in A_{\ell_u}$ (some cases are depicted in Figure 22). Hence, $u$ and $v$ are adjacent in $H(P)$, and since $H(P)$ is an interval graph by definition, $H(P)$ is chordal and a triangulation of $G$. To show that $H(P)$ is a minimal triangulation of $G$, let $uv \in E(H(P)) \setminus E$. Then, there is $j \in \{1, \dots, k-1\}$ such that $u, v \in A_j \cap A_{j+1}$, since otherwise $u$ and $v$ are adjacent in $G$ by statement (1) and Lemma 10.14. Note that there is a vertex in $A_j$ that is not contained in $A_{j+1}$ ($A_j$ contains a vertex of the enclosure of $s_j$ that is to the left of $s_j$, hence not contained in $A_{j+1}$); analogously, there is a vertex in $A_{j+1}$ that is not contained in $A_j$. Then, $uv$ is unique chord in a cycle of length 4 in $H(P)$. Hence, $H(P)$ is a minimal triangulation of $G$ due to Theorem 8.14.

■

Due to statement (1) of Lemma 10.26, $A_1, \dots, A_k$ are the maximal cliques of $H(P)$ and $\langle A_1, \dots, A_k \rangle$ is a consecutive clique arrangement for $H(P)$.

Now, we are ready to prove our main theorem about minimal triangulations of permutation graphs.

**Theorem 10.27** *Let $G = G(\pi)$ be a permutation graph over $\{1, \dots, n\}$. An interval graph $H$ on vertex set $\{1, \dots, n\}$ is a minimal triangulation of $G$ if and only if there is a maximal path $(x_0, \dots, x_k)$ in $\mathcal{PC}(\pi)$ such that $\langle A_1, \dots, A_k \rangle$ is a consecutive clique arrangement for $H$ where $A_i =_{\text{def}} V(G[s_{i-1}, s_i])$ and the special scanlines $s_{i-1}$ and $s_i$ are the labels of $x_{i-1}$ and $x_i$ in $\mathcal{PC}(\pi)$ for all $i \in \{1, \dots, k\}$.*

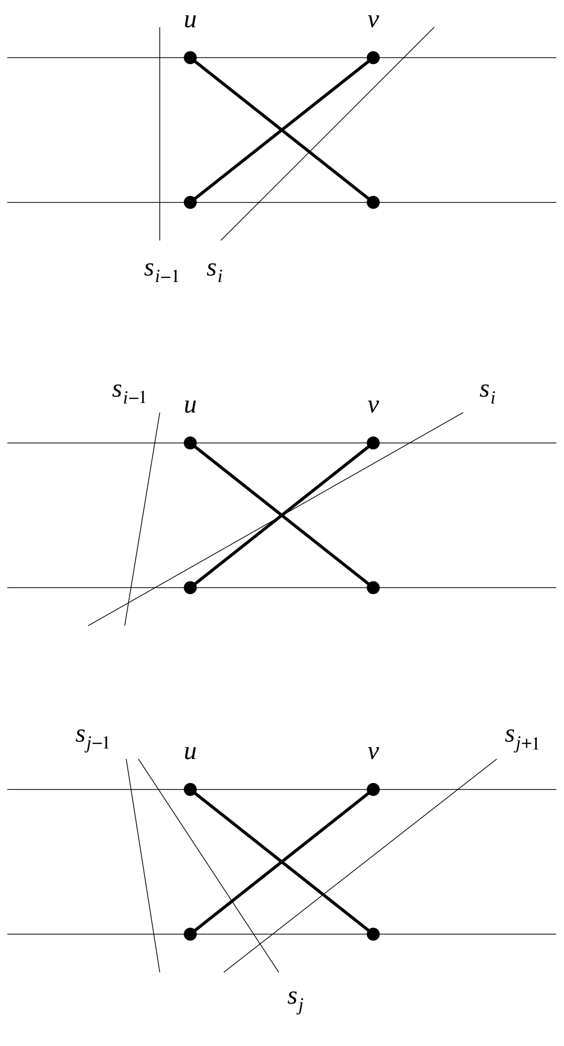


**Figure 22** Graph $H(P)$ is a triangulation of $G$. We illustrate three cases how adjacent vertices $u$ and $v$ appear in one maximal clique of $H(P)$.

**Proof:** By Lemma 10.26 and the discussions and constructions before, every maximal path of $\mathcal{PC}(\pi)$ defines a minimal triangulation of $G$ in the required sense, which proves one direction of the theorem. For the other, let $H$ be a minimal triangulation of $G$. Due to Theorem 10.22, $H$ is an interval graph, and by Theorem 10.6, $H$ has a consecutive clique arrangement $\langle A_1, \ldots, A_k \rangle$.

Let $S_0 =_{\mathrm{def}} S_k =_{\mathrm{def}} \emptyset$ and $S_i =_{\mathrm{def}} A_i \cap A_{i+1}$, $i \in \{1, \ldots, k-1\}$. By Corollary 10.7, $S_1, \ldots, S_{k-1}$ are the minimal separators of $H$, and by Theorem 8.15, $\{S_1, \ldots, S_{k-1}\}$ is a maximal set of pairwise non-crossing minimal separators of $G$. Since $A_1, \ldots, A_k$ are potential maximal cliques of $G$, there are special scanlines $s_1, s_1', s_2, \ldots, s_k'$ of $G$ such that $A_i = V(G[s_i, s_i'])$ and $s_i \in N_\pi^{\leq}(s_i')$ for $i \in \{1, \ldots, k\}$ due to Theorem 10.20. Suppose there are $i, j \in \{1, \ldots, k\}$, $i < j$, such that $s_j$ or $s_j'$ intersects with $s_i$ or $s_i'$. Since an inner vertex of the one set does not belong to the other set, $A_i$ and $A_j$ do not contain inner vertices (remember that, if, for example, $A_i$ has an inner vertex $x$ and $s_j$ intersects with $s_i'$, then $x$ is contained in $\mathrm{int}(s_j)$ and in $A_j$, which contradicts the definition of inner vertex). Hence, $1 < i < j < k$ and $A_i = S_{i-1} \cup S_i$ and $A_j = S_{j-1} \cup S_j$. By Lemma 10.19, $\{\mathrm{int}(s_i), \mathrm{int}(s_i')\} = \{S_{i-1}, S_i\}$ and $\{\mathrm{int}(s_j), \mathrm{int}(s_j')\} = \{S_{j-1}, S_j\}$. So, $s_i, s_i', s_j, s_j'$ are pairwise non-intersecting by Theorem 8.15 and Lemma 10.12. It follows that $S =_{\mathrm{def}} \{s_1, \ldots, s_k'\}$ is a set of pairwise non-intersecting special scanlines of $G$. Suppose $S$ is properly contained in a maximal set $S'$ of pairwise non-intersecting special scanlines of $G$. $S'$ uniquely defines a path in $\mathcal{PC}(\pi)$ by Lemma 10.25, and let $\mathcal{C}_{S'}$ be the set of potential maximal cliques defined by $S'$ (i.e., on the path of $\mathcal{PC}(\pi)$ defined by $S'$). It holds that $\mathcal{C}_S =_{\mathrm{def}} \{A_1, \ldots, A_k\} \subseteq \mathcal{C}_{S'}$. By Lemma 10.26, $\mathcal{C}_{S'}$ defines a minimal triangulation $H'$ of $G$, and since $\mathcal{C}_S \subseteq \mathcal{C}_{S'}$, $H$ is a subgraph of $H'$. Since $H$ is a triangulation of $G$, $H$ and $H'$ are equal, and $\mathcal{C}_S$ and $\mathcal{C}_{S'}$ are equal, and the theorem holds. It might also be the case that $S$ is a maximal set of pairwise non-intersecting special scanlines of $G$ but $\mathcal{C}_S \subset \mathcal{C}_{S'}$. We similarly conclude that this is not possible.

■

## 10.5 Solving the min-Tri membership problem

The membership problem that we wish to solve in this section is defined as follows: given a pair $(G, H)$ of graphs, where $G$ is a permutation graph, is $H$ a minimal triangulation of $G$? We will call this problem the *min-Tri membership problem for permutation graphs.* A variant of this problem is called the *promise min-Tri membership problem for permutation graphs* where we trust that the input graph $G$ is a permutation graph. The input is additionally furnished with the permutation sequence defining the input permutation graph, i.e., input is a triple $(G, \pi, H)$, where $G = G(\pi)$.

Our algorithms for solving the min-Tri membership problems for permutation graphs are based on the characterisation of minimal triangulations of permutation graphs by maximal paths of potential maximal cliques graphs (Theorem 10.27).

Therefore, a central part is dedicated to the problem of efficiently generating the potential maximal cliques graph of a given permutation graph. From its definition and Corollaries 10.9 and 10.21 we know that it may be possible to generate the potential maximal cliques graph in linear time, and Corollary 10.9 already states that the set of special scanlines can be listed in linear time. So, the major problem remains to list the set of arcs.

**Lemma 10.28** *Let $G = G(\pi)$ be a permutation graph over $\{1, \dots, n\}$ that has $m$ edges. In time $\mathcal{O}(n + m)$, the potential maximal cliques graph $\mathcal{PC}(\pi)$ of $G$ can be generated.*

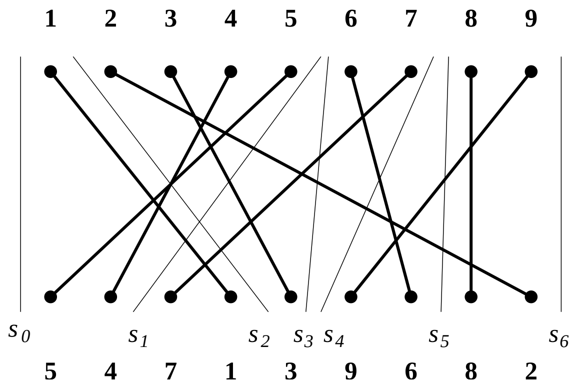
**Proof:** Due to Corollary 10.9, the set of special scanlines of $G$ can be listed in linear time, i.e., in time $\mathcal{O}(n+m)$. It is clear by Theorem 10.20 that the sets $N^{<}_{\pi}(s)$ for $s$ a special scanline of $G$, $\mathrm{s}_0 < s$, represent the arc set of $\mathcal{PC}(\pi)$. So, it suffices to show that these neighbours sets can be generated in linear time. We define an order $\rightharpoonup$ on the special scanlines of $G$ as follows: order the special scanlines increasingly by their upper endpoints and within an equivalence class by their lower endpoints. In other words, for scanlines $s = (a, e)$ and $s' = (a', e')$, $s \rightharpoonup s'$ if and only if $a < a'$ or, if $a = a'$, $e < e'$. Similarly, we define order $\rightharpoondown$: $s \rightharpoondown s'$ if and only if $e < e'$ or, if $e = e'$, $a < a'$. Note that $\rightharpoonup$ and $\rightharpoondown$ define linear orders, hence orderings, and they can be obtained in linear time. Furthermore, $\rightharpoonup$ and $\rightharpoondown$ together define a permutation graph, and a permutation diagram can be computed in linear time: on the upper line, write the special scanlines according to order $\rightharpoonup$, on the lower line, write the special scanlines according to order $\rightharpoondown$, join two points with the same scanline label. It is easy to see that, for two special scanlines $s$ and $s'$ of $G$, $s < s'$ if and only if $s \rightharpoonup s'$ and $s \rightharpoondown s'$. This also means that $s$ and $s'$ intersect if and only if either $s \rightharpoonup s'$ or $s \rightharpoondown s'$. We associate with this permutation diagram a permutation sequence and a permutation graph that is obtained by simply replacing the scanline labels by natural numbers. An example is given in Figure 23.

We define two functions $g$ and $h$. Let $g$ assign to every special scanline $s$, $\mathrm{s}_0 < s$, the rightmost special scanline with respect to $\rightharpoondown$ that does not intersect with $s$ and is to the left of $s$. In other words, $g(s) \rightharpoondown s$ and, for every special scanline $s'$ of $G$, if $g(s) \rightharpoondown s' \rightharpoondown s$, then $s'$ intersects with $s$. Note that we could have used $\rightharpoonup$ alternatively and obtained the same function $g$. For convenience, let $g(\mathrm{s}_0) =_{\mathrm{def}} \mathrm{s}_0$. By $h$, we denote the mapping that assigns to every special scanline $s$ the special scanline $s'$ that is rightmost with respect to $\rightharpoondown$ where $s' \rightharpoondown s$ and that intersects with $s$. If there is no such scanline, let $h(s) =_{\mathrm{def}} s$. We claim that, for all special scanlines $s$ of $G$, $\mathrm{s}_0 < s$, holds

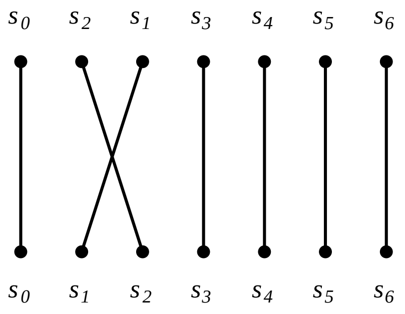
$$N^{<}_{\pi}(s) = \{h^i(g(s)) : i \geq 0\} \cap \{s' < s\}$$

where $h^0(g(s)) =_{\mathrm{def}} g(s)$ and $h^{i+1}(g(s)) =_{\mathrm{def}} h(h^i(g(s)))$. To verify the inclusion from left to right, first observe that, for special scanline $s$, the scanlines in $N^{<}_{\pi}(s)$ are pairwise intersecting, second, the element in $N^{<}_{\pi}(s)$ rightmost with respect to $\rightharpoondown$ is $g(s)$: let $N^{<}_{\pi}(s) = \{t_1, \dots, t_r\}$ where $t_r \rightharpoondown \cdots \rightharpoondown t_1$, then $h^i(g(s)) = t_{i+1}$ for

Step 1: Listing special scanlines



Step 2: Defining a permutation diagram


$s_0$ $s_2$ $s_1$ $s_3$ $s_4$ $s_5$ $s_6$
$s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$

Step 3: Constructing a permutation sequence

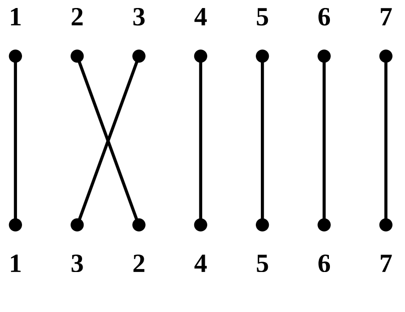


**Figure 23** For graph $G = G(\langle 5,4,7,1,3,9,6,8,2\rangle)$ we obtain the auxiliary graph $G(\langle 1,3,2,4,5,6,7\rangle)$ for generating the potential maximal cliques graph of $G$ using the algorithm of the proof of Lemma 10.28.

all $i \le r-1$. The inclusion from right to left is clear by definition of $g$ and $h$. So, if $g$ and $h$ can be computed in linear time, more precisely, if we can compute arrays representing both functions, the sets $N_\pi^{<}(s)$ can be computed in overall linear time, since $\mathcal{PC}(\pi)$ contains at most $n+2m$ arcs due to Corollary 10.21. Note that if $h^i(g(s)) \notin \{s' : s' < s\}$ then $h^{i+1}(g(s)) \notin \{s' : s' < s\}$.

From orders $\rightharpoonup$ and $\rightarrow_\tau$, we have defined a permutation graph. It is easy to see that the results of functions $h$ and $g$ then correspond to closest left neighbours and non-neighbours, respectively. Let $G(\tau)$ be a permutation graph over $\{1,\dots,k\}$. Let $x$ be a vertex of $G(\tau)$. The *closest left neighbour* of $x$ is the vertex $z$ such that $\tau^{-1}(z) < \tau^{-1}(x)$ and $x$ and $z$ are adjacent and there is no neighbour $w$ of $x$ such that $\tau^{-1}(z) < \tau^{-1}(w) < \tau^{-1}(x)$. If there is no left neighbour of $x$, we say that $x$ is its own closest left neighbour. The definition of closest left non-neighbour is similar; we only replace "adjacent" and "neighbour" by "non-adjacent" and "non-neighbour", respectively. Now, consider the following algorithm that is applied to $G(\tau)$:

(1) initialize a stack with value $k{+}1$

(2) for $i = 1,\dots,k$: delete all vertices from the stack that are smaller than $\tau(i)$, assign the top-of-stack element as closest left neighbour to $\tau(i)$, push $\tau(i)$ on the stack

(3) for $x \in \{1,\dots,k\}$: if $x$ has $k{+}1$ as closest left neighbour, assign $x$ as closest left neighbour to $x$.

The described algorithm runs in time $\mathcal{O}(k)$. For the correctness, observe that vertices that are deleted in step 2 are non-neighbours of the current vertex and that all remaining vertices in the stack (except for $k{+}1$) are left neighbours of the current vertex. It is clear that the top-of-stack element is rightmost in $\tau$ among all vertices in the stack (since it is latest pushed) and so is the closest left neighbour. For computing the closest left non-neighbour, use the same algorithm on permutation graph $G(\tau')$ where $\tau' =_{\text{def}} k+1-\tau$. The position of the closest left neighbour of $\tau'(i)$ is the position of the closest left non-neighbour of $\tau(i)$.

■

**Theorem 10.29 (Booth and Lueker, [11])**
*Let $G = (V,E)$ be a graph. In linear time, it can be verified whether $G$ is an interval graph. Furthermore, if $G$ is an interval graph, an interval model $\mathfrak{I}$ such that $G = G(\mathfrak{I})$ can be generated in linear time.*

**Theorem 10.30** *There is a linear-time algorithm for solving the promise min-Tri membership problem for permutation graphs.*

**Proof:** Let $(G,\pi,H)$ be an instance of our promise min-Tri membership problem. We can assume $G = G(\pi)$. Due to Lemma 10.28, the potential maximal cliques graph $\mathcal{PC}(\pi)$ of $G$ is computed in linear time. If $H$ is a minimal triangulation of $G$, then $H$ is an interval graph due to Corollary 10.24. In linear time, it can be verified whether $H$ is an interval graph, and if so, an appropriate interval model can be generated due to Theorem 10.29. Furthermore, Theorem 10.6 gives a linear-time algorithm for computing a consecutive clique arrangement $\langle A_1, \ldots, A_k \rangle$ for $H$ from the interval model. According to Theorem 10.27, we have to find a maximal path in $\mathcal{PC}(\pi)$ that defines set $\{A_1, \ldots, A_k\}$ as potential maximal cliques of $G$. We assume that $H$ is a triangulation of $G$.

For every vertex $x$ of $G$, let $a(x)$ denote the number of maximal cliques of $H$ containing $x$. Note that, if $k \geq 2$, there are two vertices $x$ and $x'$ of $G$ such that $a(x) = a(x') = 1$. Roughly, our algorithm works as follows: select a vertex $u$ for which holds $a(u) = 1$, find an appropriate arc in the potential maximal cliques graph of $G$, adjust the values of $a(x)$ for the still unselected vertices and repeat until there is no vertex left. The value $a(x)$ in every iteration denotes the number of maximal cliques of $H$ in which $x$ appears and whose corresponding arcs in $\mathcal{PC}(\pi)$ have not yet been found.

Remember that all arcs of $\mathcal{PC}(\pi)$ with start vertex labelled with $\mathrm{s}_0$ as well as all arcs with end vertex labelled with $\mathrm{s}_e^n$ correspond to potential maximal cliques of $G$ with inner vertex. A vertex $x$ with $a(x) = 1$ is inner vertex of a potential maximal clique of $G$ and therefore represents a unique potential maximal clique of $G$. Mark all arcs of $\mathcal{PC}(\pi)$ that correspond to maximal cliques of $H$ with inner vertex (if potential maximal cliques of $G$ do not correspond with maximal cliques of $H$, the input can be rejected). It holds that $H$ is a minimal triangulation of $G$ if and only if there is a path in $\mathcal{PC}(\pi)$ containing all already marked arcs whose arcs correspond to the maximal cliques of $H$. On such a path, unmarked arcs must correspond to potential maximal cliques of $G$ without inner vertex.

We describe an iteration step. Let $P = (z_0, \ldots, z_r)$ be the path in $\mathcal{PC}(\pi)$ that starts at the source vertex of $\mathcal{PC}(\pi)$, contains only marked arcs and is longest possible. Hence, $r \geq 1$. We know that every arc on $P$ corresponds to a maximal clique of $H$. We have to select a successor of $z_r$. If there is only one possible choice, the next vertex is chosen, the arc is marked and the values $a(x)$ are adjusted. Suppose there are two possible choices $z'$ and $z''$. Let $s$ be the scanline vertex $z_r$ is labelled with. Vertices $z'$ and $z''$ are labelled with scanlines, let them be $s'$ and $s''$, that are close neighbours of $s$. If $H$ is a minimal triangulation of $G$ either $V(G[s, s'])$ or $V(G[s, s''])$ is a maximal clique of $G$. (For instance, $s'$ and $s''$ represent crossing minimal separators.) Note that there is a vertex $x$ from the enclosure of $s'$ or $s''$ such that $a(x) = 1$. Using this vertex, the correct arc can be selected. Assume that we select $z'$. For adjusting the values $a(x)$ for every vertex $x$, it remains to consider vertices that are contained in $V(G[s, s'])$, and these vertices are all neighbours of two vertices from $V(G[s, s'])$, one from the enclosure of $s$, the other from the enclosure of $s'$. Every vertex can be in at most two such pairs, such that adjustment takes only linear time. While adjusting, vertices for which only one maximal clique remains unhandled can be found and stored in a list, so that the described algorithm becomes linear-time.

■

For solving the min-Tri membership problem for permutation graphs, it must be verified whether graph $G$ of input $(G, H)$ is a permutation graph. However, for a linear-time solution we relax this question and only require that $G$ is *isomorphic* to a permutation graph. It is not clear, given a permutation graph $G$ (in our restricted sense), whether a permutation $\pi$ can be generated in linear time such that $G = G(\pi)$.

**Theorem 10.31** *There is a linear-time algorithm for solving the (relaxed) min-Tri membership problem for permutation graphs.*

**Proof:** Let $(G, H)$ be an instance of our min-Tri membership problem. In linear time, it can be recognized whether $G$ is isomorphic to a permutation graph, and an appropriate permutation $\pi$ can be generated in linear time due to Theorem 10.2. Let $G'$ and $H'$ emerge from $G$ and $H$ by renaming the vertices according to $\pi$. It holds that $H'$ is a minimal triangulation of $G'$ if and only if $H$ is a minimal triangulation of $G$. In another linear time, it can be verified by the promise min-Tri membership problem whether $H'$ is a minimal triangulation of $G'$, i.e., whether $(G', \pi, H')$ is element of the promise min-Tri membership problem for permutation graphs.

■

## 10.6 Algorithmic applications

Using the potential maximal cliques graph of a permutation graph, we can compute treewidth and minimum fill-in in linear time. This improves the previously best known algorithms that run in times $\mathcal{O}(n \cdot \mathrm{tw}(G))$ for treewidth and $\mathcal{O}(n^2)$ for minimum fill-in [9]. We remark that the complexity for computing the bandwidth of permutation graphs is still open, i.e., it is not even known whether BANDWIDTH is polynomial-time solvable (in case of P $\neq$ NP).

We show that treewidth and minimum fill-in can be solved as special weighted shortest paths problems. Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. The *weighted* potential maximal cliques graph of $G$ is $\mathcal{PC}(\pi)$ that additionally has vertex and arc weights: vertex $x$ whose scanline label is $s$ is weighted with $|\mathrm{int}(s)|$, the cardinality of $\mathrm{int}(s)$, and arc $(x, x')$ where $s$ and $s'$ are the labels of $x$ and $x'$, respectively, is weighted with $|V(G[s, s'])|$, the cardinality of the potential maximal clique of $G$ defined by $s$ and $s'$.

**Lemma 10.32** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. The cardinalities of* int *for the special scanlines of $G$ can be computed in linear time.*

**Proof:** By Corollary 10.9, the set of special scanlines of $G$ can be listed in linear time. For every special scanline $s$ of $G$, we partition the complement of $\mathrm{int}(s)$ into two subsets and compute their cardinalities separately; the cardinality of $\mathrm{int}(s)$ is the result of an easy calculation. Let $s = (a, e)$ be a scanline of $G(\pi)$. By $\overline{\mathrm{int}}^{<}(s) =_{\mathrm{def}} \{x < a : \pi^{-1}(x) < e\}$, we denote the set of vertices of $G$ that are smaller than $a$ and that are not contained in $\mathrm{int}(s)$. By $\overline{\mathrm{int}}^{>}(s) =_{\mathrm{def}} \{x > a : \pi^{-1}(x) > e\}$, we denote the set of vertices greater than $a$ outside $\mathrm{int}(s)$. We first describe an algorithm for computing the cardinalities of $\overline{\mathrm{int}}^{<}$.

We assume that the scanlines are ordered decreasingly by the lower endpoint. (We can use order $\rightarrow_T$ of the proof of Lemma 10.28 in reversed order.) We will iteratively manipulate a list of sets, which is inspired by permutation diagrams. Let $\mathcal{L}(n)$ be an ordered list of singleton sets containing the numbers from 0 to $n$. Assign to every set the contained number as label. Iteratively, obtain $\mathcal{L}(i)$ from $\mathcal{L}(i{+}1)$, $i \in \{1, \ldots, n{-}1\}$, by the following procedure:

(a) find set $L$ in $\mathcal{L}(i{+}1)$ containing $\pi(i{+}1)$ (it will be the smallest element in that set); let $L$ have label $a$

(b) find set $L'$ in $\mathcal{L}(i{+}1)$ with the largest label smaller than $a$

(c) add the elements of $L$ to $L'$

(d) decrease by 1 the labels of all sets in $\mathcal{L}(i{+}1)$ with label greater than $a$

(e) delete (now empty set) $L$.

For every $\mathcal{L}(i)$, $i \in \{1, \ldots, n\}$, the labelling numbers of the sets are ordered increasingly and successively (remember that we have lists whose elements are ordered by their labels). It is the case that the cardinality of $\overline{\mathrm{int}}^{<}(s)$ for a scanline $s = (a{+}\frac{1}{2}, e{+}\frac{1}{2})$ is the label of the set in $\mathcal{L}(e)$ that contains $a$. Our algorithm corresponds to the following procedure on permutation diagrams: for $i \in \{1, \ldots, n\}$ in decreasing order, in step $i$ delete vertex $\pi(i)$. For a scanline $s = (a, e)$, all vertices $x$ where $\pi^{-1}(x) > e$ do not contribute to $\overline{\mathrm{int}}^{<}(s)$.

We first show that $\mathcal{L}(n), \ldots, \mathcal{L}(1)$ can be computed in linear time. We do not save every $\mathcal{L}(i)$ but manipulate on the same object. In an array, pointers to the list elements are kept to achieve constant time access to the sets by given label. Every set is kept as a chained list. Then, it is clear that steps (b) and (c) take only constant time, since $L'$ has label $a{-}1$ and computing the union in step (c) is mere pointer manipulation. However, instead of adding $L$ to $L'$, we add $L'$ to $L$. For changing the labels in step (d), pointers in the array and the labels themselves have to be adjusted. Observe that every set in $\mathcal{L}(i{+}1)$ with label greater than $a$ contains a neighbour of $\pi(i)$, which gives linear time. To provide constant time access to the label of the set containing a given number, we keep another array containing a pointer for each number from $\{1, \ldots, n\}$ to the set containing it. Steps (c) and (e) require updating these pointers, but since all but the smallest elements in $L'$ are neighbours of $\pi(i)$ (they are smaller than $\pi(i)$ but have been deleted before), we still have linear time.

To compute the cardinalities of $\overline{\mathrm{int}}^{>}$, we use the same algorithm on permutation sequence $\pi' =_{\mathrm{def}} \langle n{+}1{-}\pi(n), \ldots, n{+}1{-}\pi(1)\rangle$. Note that $G(\pi)$ and $G(\pi')$ are isomorphic, since we have only mirrored the permutation diagram of $G$. Therefore, it holds that $(a, b)$ is a special scanline of $G(\pi)$ if and only if $(n{+}1{-}a, n{+}1{-}e)$ is a special scanline of $G(\pi')$. Finally, we obtain the cardinalities of int by the following formula: $|\mathrm{int}(s)| = n - |\overline{\mathrm{int}}^{<}(s)| - |\overline{\mathrm{int}}^{>}(s)|$. ■

**Lemma 10.33** *Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$ that has $m$ edges. In time $\mathcal{O}(n+m)$, the weighted potential maximal cliques graph of $G$ can be generated.*

**Proof:** By Lemma 10.28, we can generate the potential maximal cliques graph $\mathcal{PC}(\pi)$ of $G$ in linear time. By Lemma 10.32, we can weight every vertex of $\mathcal{PC}(\pi)$ with the cardinality of the minimal separator of $G$ represented by its scanline label. It remains to add the weights to the arcs. Consider the following equivalence. Let $s_1 = (a_1, e_1)$ and $s_2 = (a_2, e_2)$ be special scanlines of $G$, $s_1 \in N_\pi^<(s_2)$, and let $C =_{\text{def}} V(G[s_1, s_2])$, $S_1 =_{\text{def}} \text{int}(s_1)$ and $S_2 =_{\text{def}} \text{int}(s_2)$. Then, we claim that

$$|C| = \frac{1}{2} \cdot \Big(|S_1| + |S_2| + (a_2 - a_1) + (e_2 - e_1)\Big) .$$

To see this equality, note the following:

- every vertex in $S_1 \cap S_2$ contributes exactly 2 to $|S_1| + |S_2|$
- every vertex in $C \setminus (S_1 \cup S_2)$ contributes exactly 2 to $(a_2 - a_1) + (e_2 - e_1)$
- every vertex in $(S_1 \cup S_2) \setminus (S_1 \cap S_2)$ contributes exactly 1 to $|S_1| + |S_2|$ and to $(a_2 - a_1) + (e_2 - e_1)$
- vertices not contained in $C$ do not contribute to the sum.

Hence, from the labels and weights of the start and end vertex of every arc, its weight can be computed in constant time. Since $\mathcal{PC}(\pi)$ contains at most $n + 2m$ arcs, we obtain a linear-time algorithm for computing the weighted potential maximal cliques graph of $G$.

■

**Theorem 10.34** *Treewidth and minimum fill-in for permutation graphs can be computed in linear time.*

**Proof:** Let $G = G(\pi)$ be a permutation graph over $\{1, \ldots, n\}$. Let $\mathcal{W}$ be the weighted potential maximal cliques graph of $G$. Due to Lemma 10.25, $\mathcal{W}$ is acyclic and therefore has a topological ordering. Observe that order $\rightarrow_\tau$ of the proof of Lemma 10.28 defines such a topological ordering, that can be generated in linear time.

For computing the treewidth of $G$, it suffices to find a minimal triangulation of $G$ with smallest clique number among all minimal triangulations of $G$ due to Lemma 8.21. The clique number of a graph is the maximal size among its maximal cliques. So, applying Theorem 10.27, we have to find a maximal path in $\mathcal{W}$ whose largest arc weight is smallest among all maximal paths in $\mathcal{W}$. Such a path can be obtained by dynamic programming and processing the vertices of $\mathcal{W}$ according to $\rightarrow_\tau$ in the following way. During the process, every vertex of $\mathcal{W}$ is labelled with a number that shall represent the smallest maximal arc weight among all paths of $\mathcal{W}$ that start at the source vertex and end in that vertex. We label the source vertex, the vertex labelled with scanline $\mathrm{s}_0$, with number 0. For a vertex $x$ of $\mathcal{W}$, let $x_1, \ldots, x_r$ be its predecessors and let $a_1, \ldots, a_r$ be their number labels, respectively. Since we process the vertices according to $\rightharpoondown$, these number labels have already been computed. Furthermore, let $b_1, \ldots, b_r$ denote the weights of arcs $(x_1, x), \ldots, (x_r, x)$. Then,

$$\min\{\max\{a_i, b_i\} : 1 \leq i \leq r\}$$

is the label of $x$. We obtain that the number label of the sink vertex of $\mathcal{W}$, the vertex that is labelled with scanline $\mathrm{s}_e^n$, is the clique size of a minimal triangulation of $G$ that has smallest clique size among all minimal triangulations of $G$.

For computing the minimum fill-in, we can proceed in a fashion similar to computing the treewidth. Consider the following observation. Let $A_1, \ldots, A_k$ be a consecutive clique arrangement of an interval graph $H$. For every $i \in \{1, \ldots, k\}$:

$$|E(H[A_0 \cup \cdots \cup A_i])| + |E(H[A_{i-1} \cap A_i])| = |E(H[A_0 \cup \cdots \cup A_{i-1}])| + |E(H[A_i])|,$$

where $A_0 =_{\text{def}} \emptyset$. Since $A_{i-1} \cap A_i$ is a minimal separator of $H$ (except for $i = 1$, of course) due to Corollary 10.7, $H[A_{i-1} \cap A_i]$ is complete. So, if the numbers of vertices of $H[A_{i-1} \cap A_i]$ and $H[A_i]$ are known, it is easy to determine the numbers of edges of these graphs. Hence, a path of $\mathcal{PC}(\pi)$ that represents a minimal triangulation of $G$ with the smallest number of edges among all minimal triangulations of $G$ can be found in the following way. We label every vertex with a number. We label the source vertex of $\mathcal{W}$ with number 0. For a vertex $x$ of $\mathcal{W}$, let $x_1, \ldots, x_r$ be its predecessors and let $a_1, \ldots, a_r$ be their number labels, respectively. Furthermore, let $b_1, \ldots, b_r$ denote the weights of arcs $(x_1, x), \ldots, (x_r, x)$, respectively, and let $c_1, \ldots, c_r$ denote the weights of the vertices $x_1, \ldots, x_r$, respectively. Then,

$$\min\left\{a_i + \binom{b_i}{2} - \binom{c_i}{2} : 1 \leq i \leq r\right\}$$

is the label of $x$. Similar to treewidth above, it follows that the number label of the sink vertex of $\mathcal{W}$ gives the smallest number of edges in a minimal triangulation of $G$, and the minimum fill-in then is the difference between this number and the number of edges of $G$.

■

By the definition of the algorithms for computing treewidth and minimum fill-in, we can even compute minimal triangulations of the given permutation graph that realize its treewidth and minimum fill-in by saving the chosen predecessor, which defines an appropriate maximal path. Due to Theorem 10.27, we obtain consecutive clique arrangements for the minimal triangulations, from which it is easy to obtain interval models.

## 10.7 Interesting problems

Bodlaender, Kloks, Kratsch used the scanline approach to compute treewidth and minimum fill-in of permutation graphs [8]. This method was extended by Bodlaender, Kloks, Kratsch, Müller to solve the treewidth and minimum fill-in problems for $d$-trapezoid graphs [9]. A $d$-trapezoid graph has a geometric model that is a generalisation of permutation diagrams. On each of $d+1$ horizontal lines mark $2n$ points and label these points with the numbers from 1 to $n$ so that each number appears twice. Then, on every line and for every $x \in \{1, \ldots, n\}$, there is a left and a right occurrence of $x$. Join the points on neighbouring lines with the same label (number and "left" or "right") by line segments. The lines joining the left occurrences of $x$ and the lines joining the right occurrences of $x$ enclose a figure, that can be considered the union of $d$ trapezoids. It was shown that minimal separators of $d$-trapezoid graphs can be obtained by scanlines [9]. Hence, it is natural to ask whether our approach for solving the treewidth and minimum fill-in problems for permutation graphs can be extended to improve previous results for $d$-trapezoid graphs.

# Chapter 11
# AT-free claw-free graphs

We have studied minimal triangulations of permutation graphs. The class of permutation graphs is a subclass of the class of AT-free graphs. In this chapter, another subclass of AT-free graphs becomes the centre of our interest. The class of AT-free claw-free graphs is the set of graphs that are AT-free and do not contain the claw as induced subgraph. So, the class of AT-free claw-free graphs is defined by combining a structural with a forbidden induced subgraph property. It holds that the class of permutation graphs is not contained in the class of AT-free claw-free graphs, and vice versa.

Similar to previous chapters, we will give a characterisation of minimal triangulations of AT-free claw-free graphs. This characterisation will be based on special vertex orderings. These orderings will represent minimal triangulations, and they are generated by an algorithm, that is based on a special lexicographic breadth first search strategy inspired by `LexBFS`. This algorithm is called `min-LexBFS`. A considerable part of this chapter is dedicated to the study of properties of orderings that are generated by `min-LexBFS`. Similar to `LexBFS`, `min-LexBFS` always numbers last the vertices of a moplex of the input graph. This moplex property of vertex orderings generated by `min-LexBFS` is a strong structural result and of great help when proving other results about these orderings.

Another result, that appears only as auxiliary, is a simple and special recognition algorithm for proper interval graphs. Proper interval graphs are interval graphs, and they have interval models without intervals that are properly contained in other intervals. Proper interval graphs are related to AT-free claw-free graphs as interval graphs are related to AT-free graphs, i.e., proper interval graphs are the chordal AT-free claw-free graphs. A number of proper interval graph recognition algorithms is known from the literature, but most of them are a bit technical. Only recently and independently, new recognition algorithms were developed that are based on lexicographic breadth first search and therefore become somewhat "nicer". Our algorithm additionally belongs to a new class of recognition algorithms, namely *certifying* algorithms. The main idea is that the algorithm proves its decision: in case of acceptance as well as rejection. Most recognition algorithms prove only one of the two possible decisions. Certifying algorithms are more reliable. Our certifying proper interval graph recognition algorithm is part of the algorithm for solving the min-Tri membership problem for AT-free claw-free graphs.

## 11.1 The `min-LexBFS` algorithm

This section is dedicated to presenting the `min-LexBFS` algorithm and discussing important properties. The algorithm works on graphs and generates vertex orderings. An early version of `min-LexBFS` was introduced in [63], but this algorithm worked only on families of finite sets. Nevertheless, already the early version of `min-LexBFS` showed connections to the well-known `LexBFS` algorithm introduced by Rose, Tarjan, Lueker [76]. This connection inspired the final design of `min-LexBFS` as well as its name.

Algorithm `min-LexBFS` is defined as a partition refinement algorithm, which also guarantees an efficient implementation. Formally, it works on a list of vertex sets, selects single vertices and partitions sets in iteration steps. We need some definitions. Let $L = \langle C_1, \ldots, C_k \rangle$ be a list of non-empty sets. The elements of $L$ are called *boxes*. We say that box $C_i$ is *to the left of* box $C_j$, denoted as $C_i \prec C_j$, if and only if $i < j$. To obtain a new list from $L$ by partitioning, we *partition* a box $C_i$ by some rule and obtain $C'$ and $C''$, define $C'$ to be to the left of $C''$, call $C'$ the *left partition box*, $C''$ the *right partition box*, and obtain the new list $\langle C_1, \ldots, C_{i-1}, C', C'', C_{i+1}, \ldots, C_k \rangle$. This operation will shortly be call *partitioning* $C_i$ *into* $\langle C', C'' \rangle$ and denoted by $L : C_i \leftarrow \langle C', C'' \rangle$. Each such list contains only non-empty boxes, which means that empty boxes, that may appear as left or right partition box, are instantly removed. The *rightmost* box in $L$ is the box that is not to the left of any other box. Algorithm `min-LexBFS` uses this partition operation; it is presented in Figure 24. As defined above, empty boxes, that may appear after selecting the last vertex in the rightmost box or after partitioning, are removed from the list. This means list $L$ of `min-LexBFS` always contains only non-empty boxes. If, for some selected vertex $u$, the two partition boxes are non-empty, we say that $u$ *separates* $x$ and $z$ for every $x \in C \setminus N_G(u)$ and $z \in C \cap N_G(u)$. We remark that `LexBFS` is obtained from `min-LexBFS` by partitioning **every** box instead of only the leftmost box containing a neighbour (see also [76] and [37]).

As an example for the work of `min-LexBFS` on graphs, consider the graph depicted on the left side of Figure 25. The first few iteration steps of a `min-LexBFS` run on this graph are documented on the right side of Figure 25. In each step, the upright arrow points to the currently chosen vertex, and the capital letters represent its still unselected neighbours. To the right of the upright line, the final sequence is iteratively generated. Both the lists to the left and to the right of the line are ordered from left to right. The next chosen vertex will be $f$. The resulting vertex ordering will be $\langle h, i, j, g, e, f, d, c, b, a \rangle$. In fact, every min-LexBFS-ordering of the pictured graph that ends with vertex $a$ starts with the sequence $\langle h, i, j \rangle$, which can be verified by trying all possible runs. In the example, vertex $b$ separates $g$ and $e$.

**Definition 45** *Let $G = (V, E)$ be a graph. A **min-LexBFS-ordering** for $G$ is a vertex ordering $\sigma$ of the vertices of $G$ that can be generated by* `min-LexBFS` *on input $G$.*

Similar to the definition of min-LexBFS-orderings we define *LexBFS-orderings*, i.e., vertex orderings that can be generated by `LexBFS`. It holds that the sets of LexBFS- and min-LexBFS-orderings for a graph may differ in the sense that each set contains orderings that are not contained in the other set. An example is already

```
min-LexBFS($G$) returns $\sigma$:
1   begin
2       $L$ := $\langle V(G) \rangle$;
3       for $i$ := $|V(G)|$ downto 1 do
4           pick an arbitrary vertex $u$ from the rightmost box of $L$;
5           $\sigma(i)$ := $u$;
6           let $C$ be the leftmost box in $L$ containing a neighbour of $u$;
7           $L : C \leftarrow \langle C \setminus N_G(u)\, ,\, C \cap N_G(u) \rangle$
8       end for
9   end.
```

**Figure 24** The min-LexBFS algorithm.

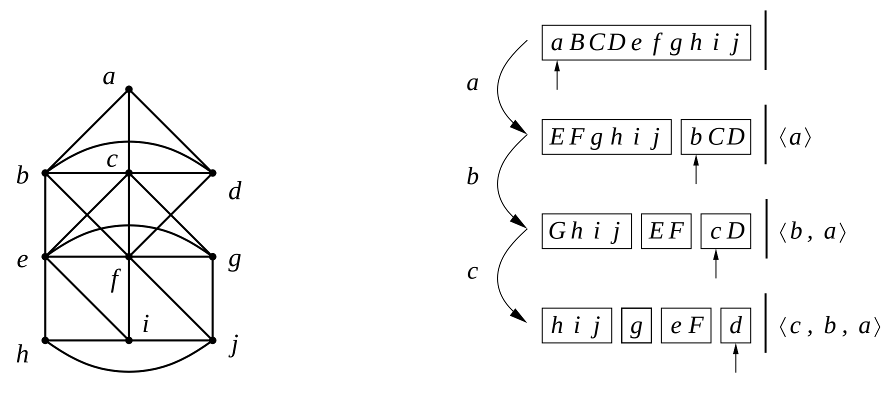

a
b
c
d
e
f
g
h
i
j
a
b
c
$\boxed{a\,B\,C\,D\,e\,f\,g\,h\,i\,j}$
$\boxed{E\,F\,g\,h\,i\,j}\ \boxed{b\,C\,D}$ $\langle a \rangle$
$\boxed{G\,h\,i\,j}\ \boxed{E\,F}\ \boxed{c\,D}$ $\langle b, a \rangle$
$\boxed{h\,i\,j}\ \boxed{g}\ \boxed{e\,F}\ \boxed{d}$ $\langle c, b, a \rangle$

**Figure 25** A graph and the beginning of a min-LexBFS run on it.

provided by the graph in Figure 25. Every LexBFS-ordering for this graph that ends with vertex $a$ starts with $\langle h, j, i\rangle$. For `min-LexBFS` and `LexBFS` runs on input graphs, we also speak of *sweeps*. So, every `min-LexBFS` sweep on the graph in Figure 25 that starts with $a$ generates a sequence with $\langle h, i, j\rangle$ as the beginning, and every `LexBFS` sweep on this graph produces only sequences with $\langle h, j, i\rangle$ as the beginning, if the sweep starts with $a$.

As we have seen in the example above, the choice of a vertex by `min-LexBFS` may not be unique. This is the case, if the rightmost box contains more than one vertex. We take care of this effect and avoid this *nondeterministic* behaviour by the following definition. Let `min-LexBFS`$^*$ be a variant of `min-LexBFS` that gets as input a graph $G$ and a vertex ordering $\sigma$ for $G$ and that works like `min-LexBFS` but always chooses that vertex in the rightmost box that is leftmost in $\sigma$. We could say that the chosen vertex has *highest preference* with respect to $\sigma$ among the vertices in the current rightmost box. Using `min-LexBFS`$^*$, we define two multi-sweep algorithm classes, that differ only in the way the first sweep is performed. Let $G$ be a graph on the vertices $1, \ldots, n$, and let $k \geq 0$:

$$\begin{aligned} 0\texttt{min-LexBFS}^*(G, \sigma) &=_{\text{def}} \sigma \\ (k{+}1)\texttt{min-LexBFS}^*(G, \sigma) &=_{\text{def}} \texttt{min-LexBFS}^*(G, k\texttt{min-LexBFS}^*(G, \sigma))\,. \end{aligned}$$

Furthermore, for $k \geq 1$:

$$\begin{aligned} k\texttt{min-LexBFS}(G) &=_{\text{def}} k\texttt{min-LexBFS}^*(G, \langle 1, \ldots, n\rangle) \\ k\texttt{free-min-LexBFS}(G) &=_{\text{def}} (k{-}1)\texttt{min-LexBFS}^*(G, \texttt{min-LexBFS}(G))\,. \end{aligned}$$

For $k \geq 1$, orderings that can be generated by $k$`min-LexBFS` are called *$k$min-LexBFS-orderings*. Similarly, *$k$free-min-LexBFS-orderings* are defined. It holds that 1free-min-LexBFS-orderings are just min-LexBFS-orderings. In contrast, $k$min-LexBFS-orderings are special min-LexBFS-orderings, namely those that are generated by `min-LexBFS` sweeps where vertices are chosen with respect to $(k{-}1)$min-LexBFS-orderings. It is clear that every graph has exactly one $k$min-LexBFS-ordering. In contrast, $k$free-min-LexBFS-orderings are not unique, and every $k$min-LexBFS-ordering is a $k$free-min-LexBFS-ordering.

**Proposition 11.1 (Meister, [63])**
*For every $k \geq 1$, algorithm $k$`min-LexBFS` can be implemented to run in linear time.*

**Proof:** It is clear that it suffices to give a linear-time implementation of `min-LexBFS`$^*$. Generally, we use the implementation that was presented in [76] for `LexBFS`. Each box is represented by a chained list and is attached to a so-called set header cell. These cells are kept in a chained list to establish the ordering of the boxes. Since we want to partition at most one box each time, we have to know which is the leftmost one. Each set header cell (each being the head of a box) is additionally furnished with two numbers: the total number of elements (vertices) in the boxes to its left and the number of elements in its attached box. The box to be partitioned has the smallest total number of vertices in boxes to its left. The partition operation has to create a new box and update two numbers. For every selected vertex, this takes time linear in its degree, which adds up to linear time in total. It remains to find the vertex with highest preference in the rightmost box. A thorough analysis of the implementation shows that, after partitioning a box, the vertices in the right partition box appear in the same order as they are kept in the adjacency list of the vertex that caused the partition (i.e., of the currently chosen vertex). In linear time, we can order the vertices in the adjacency lists of $G$ and the initializing box $V$. Then, the first vertex in the list corresponding to the rightmost box is always the one with highest preference.

■

We want to present a useful property of min-LexBFS-ordering and give a characterisation of these orderings. Let $G = (V, E)$ be a connected graph, and let $z$ be a vertex of $G$. A vertex ordering $\sigma$ for $G$ is a *BFS-ordering with root vertex z* for $G$ if, for every pair $u, v$ of vertices of $G$: $u \prec_\sigma v \Rightarrow \text{dist}(z, u) \geq \text{dist}(z, v)$. A *BFS-ordering* for $G$ is a BFS-ordering with root vertex some vertex of $G$. The *BFS-levels* of $G$ with respect to $z$ are defined by the partition $\langle S_0, \ldots, S_k \rangle$ of $G$ where $S_i$ contains all vertices at distance $i$ to $z$.

**Lemma 11.2** *Let $G = (V, E)$ be a connected graph, and let $\sigma$ be a min-LexBFS-ordering for $G$. Then, $\sigma$ is a BFS-ordering for $G$.*

**Proof:** Suppose $\sigma$ is not a BFS-ordering for $G$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with respect to $\sigma(n)$. Then, there are two vertices $u, v$ such that $u \prec_\sigma v$ and $u \in S_i$ and $v \in S_j$ and $i < j$. Without loss of generality, we assume $u$ to be rightmost among all those vertices fulfilling this criterion. When the rightmost neighbour $w$ of $u$ with respect to $\sigma$ is chosen by `min-LexBFS`, $u$ and $v$ are both contained in the leftmost box. This vertex separates $v$ and $u$, since $v$ is non-adjacent with $w$ in $G$. Hence, $v$ is in the left partition box and $u$ is in the right partition box, which contradicts the assumption.

■

Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. For every vertex $x$ of $G$, by $\text{lm}_{G,\sigma}(x)$ we denote the neighbour of $x$ in $G$ that is leftmost with respect to $\sigma$. If $x$ does not have any left neighbour $\text{lm}_{G,\sigma}(x) =_{\text{def}} x$. The indices $G$ and $\sigma$ are added only if the context requires specification.

**Definition 46** *Let $G = (V, E)$ be a graph. A vertex ordering $\sigma$ of $G$ has the* ***farther right neighbour property*** *if and only if, for every pair $u, v$ of non-adjacent vertices of $G$, the following holds: if $\text{lm}(v) \prec_\sigma u \prec_\sigma v$ then there is a neighbour $w$ of $u$ to the right of $v$ such that $\text{lm}(v) \prec_\sigma \text{lm}(w)$.*

It is interesting to observe that interval orderings have the farther right neighbour property, since the premise is always false. Hence, there are graphs that have vertex orderings with the farther right neighbour property. We can even show a stronger result: every graph has a vertex ordering with the farther right neighbour property. And we can show an even stronger result: `min-LexBFS` exactly generates these orderings.

**Theorem 11.3** *Let $G = (V, E)$ be a graph. A vertex ordering $\sigma$ for $G$ is a min-LexBFS-ordering for $G$ if and only if $\sigma$ has the farther right neighbour property.*

**Proof:** Let $\sigma$ be a min-LexBFS-ordering for $G$. Consider the generation of $\sigma$. Let $u$ and $v$ be non-adjacent vertices, $\mathrm{lm}(v) \prec_\sigma u \prec_\sigma v$. Then, $\mathrm{lm}(v)$ and $u$ cannot be contained in the same box when $v$ is chosen by `min-LexBFS`. Otherwise, $v$ separates $u$ and $\mathrm{lm}(v)$ which contradicts $\mathrm{lm}(v) \prec_\sigma u$. Hence, there is a neighbour $w$ of $u$, $v \prec_\sigma w$, that separates $\mathrm{lm}(v)$ and $u$. Obviously, $\mathrm{lm}(w)$ is in the right partition box, like $u$. Therefore, $\mathrm{lm}(v) \prec_\sigma \mathrm{lm}(w)$.

For the converse, let $\sigma$ be a vertex ordering for $G$ with the farther right neighbour property. Suppose that $\sigma$ cannot be generated by `min-LexBFS`. Then, there is a first vertex $v$ that separates two vertices $u$ and $x$, i.e., $u$ is in the left, $x$ is in the right partition box, and $x \prec_\sigma u$. In particular, $x$ is a neighbour of $v$ and $u$ is not a neighbour of $v$. Without loss of generality, we assume $x = \mathrm{lm}(v)$. Obviously, $u$ cannot be chosen before $x$ by `min-LexBFS`. By the farther right neighbour property, there is a neighbour $w$ of $u$, $v \prec_\sigma w$, that is not a neighbour of $x$. Since $u$ and $x$ are contained in one box when $w$ is chosen ($v$ separates $u$ and $x$, and $w$ is chosen before $v$), $w$ separates $x$ and $u$. This contradicts the assumption.

■

## 11.2 Moplexes and the `min-LexBFS` algorithm

Berry and Bordat introduced the notion of a moplex. A moplex is a possible generalisation (and simultaneously restriction) of a simplicial vertex. Dirac showed that every non-complete chordal graph contains two non-adjacent simplicial vertices (Theorem 8.8). Berry and Bordat showed that every `LexBFS` run numbers the vertices of a moplex last [5]. It follows that every non-complete graph has two non-adjacent moplexes. This generalises Dirac's theorem. Thus, moplexes provide structural insight into the properties of LexBFS-orderings. In this section, we will see that `min-LexBFS` also numbers last the vertices of a moplex. This is a non-trivial result with respect to the work of Berry and Bordat. The definition of moplexes relies on special modules.

**Definition 47** *Let $G = (V, E)$ be a graph. The vertex set $M \subseteq V$ is a **module** of $G$ if and only if for all pairs $u, v$ of vertices in $M$: $N_G(u) \setminus M = N_G(v) \setminus M$.*

In other words, the vertices of a module cannot be distinguished from outside the module, since they have same neighbourhoods. For example, connected components of a graph or its complement are modules. Since every vertex itself and the whole vertex set are modules, every graph has modules. Modules containing the whole vertex set are called *trivial.* (Sometimes, modules of size at most 1 are also called trivial.) A module $M$ of a graph $G$ is a *maximal* module if it is not contained in any other module. A module is *connected* if it induces a connected graph. Modules were discovered independently several times. Gallai showed that each connected graph whose complement is also connected can uniquely be partitioned into its maximal non-trivial modules [30]. The iterative process on the maximal modules is called a modular decomposition. (Here, connected components of the graph or its complement also have to be treated adequately.) Further information about modules and modular decomposition can be found in [14] and [60].

**Definition 48** *Let $G = (V, E)$ be a graph. The vertex set $M \subseteq V$ of $G$ is a* ***clique-module*** *of $G$ if and only if $M$ is a module as well as a clique of $G$. If $M$ is not properly contained in another clique-module of $G$, $M$ is a* ***maximal*** *clique-module of $G$. $M$ is a* ***moplex*** *of $G$ if and only if $M$ is a maximal clique-module of $G$ and its neighbourhood $N_G(M)$ is a minimal separator of $G$.*

For simplicity, if $G$ is complete, the whole vertex set $V$ is a moplex of $G$ (and the only one). We call a vertex a *moplex vertex* if it belongs to a moplex. A moplex is *simplicial* if its neighbourhood is a clique. Then, a graph is chordal if and only if one can repeatedly remove simplicial moplexes from the graph until the graph is empty [5].

**Fact 11.4 (Roberts, [73])**
*Let $G = (V, E)$ be a graph, and let $M_1$ and $M_2$ be maximal clique-modules of $G$. Then, $M_1$ and $M_2$ are either equal or disjoint.*

**Proof:** Let $x$ and $z$ be two vertices of a clique-module of $G$. Then, $N_G[x] = N_G[z]$, since $x$ and $z$ have the same (closed) neighbours inside the module and outside the module. Suppose $M_1$ and $M_2$ are not disjoint. Let $u \in M_1 \cap M_2$, and let $v \in M_1$. It holds that $N_G[u] = N_G[v]$, therefore $N_G[w] = N_G[v]$ for every $w \in M_2$. Hence, $M_2 \cup \{v\}$ is a clique-module properly containing $M_2$, which is not possible due to maximality of $M_2$. It follows that $M_1 \subseteq M_2$, and equality holds by maximality of $M_1$.

■

Two moplexes are *adjacent* if one moplex contains a neighbour of the other; the union of two adjacent moplexes forms a clique. We give a new characterisation of chordal graphs by means of moplexes.

**Fact 11.5** *A graph is chordal if and only if there is no induced subgraph of $G$ that contains two adjacent moplexes.*

**Proof:** In a chordless cycle of length at least 4 every maximal clique is the union of two adjacent moplexes. Conversely, since a graph is chordal if and only if every induced subgraph of it is chordal, it suffices to show that no chordal graph contains adjacent moplexes. Let $G = (V, E)$ be a chordal graph. Suppose $M_1$ and $M_2$ are adjacent moplexes of $G$. It holds that $M_1 \subseteq N_G(M_2)$. Since $N_G(M_2)$ is a minimal separator of $G$, there are two $N_G(M_2)$-full components in $G$, and every vertex in $M_1$ is adjacent with two non-adjacent vertices. Hence, $N_G(M_1)$ is not a clique in $G$, which contradicts Theorem 8.4.

■

In the following, we will show that the first vertex of every min-LexBFS-ordering belongs to a moplex, and all vertices of this moplex are generated consecutively. As mentioned above the same property holds for LexBFS-orderings [5], and `LexBFS` has been so far the only algorithm known with this property. Our proof uses ideas that are similar to the ones used by Berry and Bordat for showing their result, but both proofs also differ in their final execution.

**Lemma 11.6** *Let $G = (V, E)$ be a connected graph, and let $\sigma$ be a min-LexBFS-ordering for $G$. Let $a =_{\text{def}} \sigma(1)$ and $z =_{\text{def}} \sigma(n)$.*

*(1) Let $M = \{a, \sigma(2), \ldots, \sigma(i)\}$, $i \in \{1, \ldots, n-1\}$, be a connected module of $G$. Then, the ordering $\sigma' = \langle a, \sigma(2), \ldots, \sigma(i)\rangle$ is a min-LexBFS-ordering for $G' =_{\text{def}} G[M]$.*

*(2) Let $G{-}z$ be disconnected, and let $C$ be a connected component of $G{-}z$. Then, $\sigma'$ originating from $\sigma$ by restriction to the vertices of $C{+}z$ is a min-LexBFS-ordering for $G' =_{\text{def}} C{+}z$.*

*(3) Let $v$ be a vertex, $a \prec_\sigma v$, such that for every vertex $w$ to the right of $v$ holds: $w \in N_G(a) \Leftrightarrow w \in N_G(v)$. Then, $M = \{a, \sigma(2), \ldots, v\}$ is a module of $G$.*

*(4) Let $P$ be a $u,z$-path, $a \prec_\sigma u$, not containing a neighbour of $a$ except for $u$. Then, there is a $u,z$-path $P' = (u, u_1, \ldots, u_r, z)$ such that no vertex of $P'$ except for $u$ is a neighbour of $a$ and $u \prec_\sigma u_1 \prec_\sigma \cdots \prec_\sigma u_r \prec_\sigma z$.*

**Proof:** (1–2) Let $u, v \in V(G')$, $uv \notin E$, and $\text{lm}_{G',\sigma'}(v) \prec_{\sigma'} u \prec_{\sigma'} v$. Obviously, $\text{lm}_{G',\sigma'}(v) = \text{lm}_{G,\sigma}(v)$. Since $\sigma$ is a min-LexBFS-ordering, i.e., it has the farther right neighbour property due to Theorem 11.3, there is a neighbour $w$ of $u$ in $G$ to the right of $v$ with respect to $\sigma$ that is not a neighbour of $\text{lm}(v)$. Since $M$ is a module, every neighbour of $u$ to the right of $\sigma(i)$ is a neighbour of $\text{lm}(v)$, too. Hence, $w \in M$, which proves (1). In case of (2), all neighbours of $u$ in $G$ are also contained in $G'$, so $w$ is contained in $G'$ and to the right of $v$.

(3) Suppose there is a vertex $u$ between $a$ and $v$ that has a neighbour to the right of $v$ that is not a neighbour of $v$ (and $a$). Then, $v$ has a neighbour to its right (the farther right neighbour property of $\sigma$) that is not a neighbour of $a$ which contradicts the premise of the statement. If $v$ has a neighbour $x$ to its right that is not a neighbour of a vertex $u$, $a \prec_\sigma u \prec_\sigma v$, $u$ has a neighbour $w$ to the right of $x$ that is not a neighbour of $a$ and therefore of $v$. Then, $v$ has a neighbour to the right of $w$ that is not a neighbour of $a$ either, which is a contradiction.

(4) Let $x$ be the rightmost neighbour of $a$. All vertices $u$ that are not to the left of $x$ have such a desired $u, z$-path, since $\sigma$ is a BFS-ordering due to Lemma 11.2. Suppose there is $u \prec_\sigma x$ such that there is a $u, z$-path $P$ not containing a neighbour of $a$ except for $u$, but no such path is of the desired form; we choose $u$ rightmost and assume $P$ to be chordless. By the choice of $u$, the neighbour $v$ of $u$ in $P$ is to the left of $u$, and there are two adjacent vertices $b, c$ in $P$ such that $b \prec_\sigma u \prec_\sigma c$. By the farther right neighbour property of $\sigma$ and $uc \notin E$, there is a neighbour $w$ of $u$ such that $a \prec_\sigma \mathrm{lm}(c) \prec_\sigma \mathrm{lm}(w) \prec_\sigma c \prec_\sigma w$. By assumption, there is a $w, z$-path of the desired form. Hence, there is a $u, z$-path $P'$ of the desired form.

■

If a $u, z$-path in $G$ is of the form of $P'$, which means that starting at $u$ we always go to a right neighbour with regard to $\sigma$, we will call it a *right-monotone* path. Of course, if we loosen the restriction of not passing neighbours of $a$ these paths trivially exist by BFS-properties. From (3), we conclude that all vertices of a module containing $a$ are numbered consecutively by `min-LexBFS`.

To prove that the first vertex of a min-LexBFS-ordering belongs to a moplex, we also have to identify a minimal separator. Remember that, for a graph $G$, a set $S$ of vertices of $G$ is a minimal separator if and only if there are two $S$-full components of $G$. Note that if $G$ is not connected then $S = \emptyset$ is a minimal separator of $G$. By definition, every separator contains a minimal separator.

**Lemma 11.7** *Let $G = (V, E)$ be a graph, and let $M \subseteq V$ be a non-trivial module of $G$. Let $S \subseteq M$ be a minimal separator of $G[M]$. Then, $S \cup N_G(M)$ is a minimal separator of $G$.*

**Proof:** Let $S$ be a minimal separator of $G[M]$. Then, there are two $S$-full components in $G[M]$ induced by $C$ and $D$. Since $N_G(C) \setminus M \subseteq N_G(M)$ and $N_G(D) \setminus M \subseteq N_G(M)$, $C$ and $D$ are connected components of $G \setminus (S \cup N_G(M))$. Furthermore, it is clear that every vertex in $N_G(M)$ has a neighbour in $C$ and in $D$, so that every vertex in $S \cup N_G(M)$ has a neighbour in $C$ and in $D$, i.e., $C$ and $D$ are $(S \cup N_G(M))$-full components in $G$, and $S \cup N_G(M)$ is a minimal separator of $G$.

■

Having proved Lemmata 11.6 and 11.7, we can prove the main theorem of this section. The *size* of a moplex means the number of vertices in the moplex.

**Theorem 11.8** *Let $G = (V, E)$ be a connected graph, and let $\sigma$ be a min-LexBFS-ordering for $G$. Then, there is $r \geq 1$ such that $M = \{\sigma(1), \ldots, \sigma(r)\}$ is a moplex of $G$ of size $r$.*

**Proof:** We will prove the theorem by induction over the number of vertices of a graph. The statement is true for all connected graphs on at most three vertices. Furthermore, it is true for all complete graphs. Let $n \geq 4$, and let the statement be true for all graphs on at most $n{-}1$ vertices. Let $G = (V, E)$ be a connected graph on $n$ vertices that is not complete, and let $\sigma$ be a min-LexBFS-ordering for $G$. Let $a =_{\text{def}} \sigma(1)$ and $z =_{\text{def}} \sigma(n)$. Let $C$ be the connected component of $G{-}z$ that contains $a$. If $az \notin E$ and $G{-}z$ is disconnected, let $G' =_{\text{def}} C{+}z$. Then, every subset of $N_G(a)$ is a minimal separator of $G$ if and only if it is a minimal separator of $G'$. If $az \in E$, then $z$ is universal. Set $S \subseteq V \setminus \{z\}$ is a minimal separator of $G{-}z$ if and only if $S \cup \{z\}$ is a minimal separator of $G$. To both cases, we apply Lemma 11.6 and conclude due to induction hypothesis. Note that the case $z$ universal and $G{-}z$ connected immediately holds by induction hypothesis.

Now, let $z$ be not universal, i.e., $az \notin E$, and let $G{-}z$ be connected. Let $v$ be the rightmost vertex with respect to $\sigma$ fulfilling $N_G[a] = N_G[v]$. Note that $a$ and $v$ are equal or adjacent and $a = \text{lm}(v)$. Let $M =_{\text{def}} \{a, \sigma(2), \ldots, v\}$. Then, $M$ is a module by Lemma 11.6. If, for some $u$, $a \prec_\sigma u \prec_\sigma v$, $u$ is not adjacent with $v$ there must be a neighbour of $u$ to the right of $v$ that is not a neighbour of $a$, contradicting $M$ being a module. Hence, $uv \in E$ and $au \in E$ for every $u \prec_\sigma v$. If there are two non-adjacent vertices $u, w \in M$, $a \prec_\sigma u \prec_\sigma w \prec_\sigma v$, $u$ must have a neighbour $x$ to the right of $w$ that is not a neighbour of $a$. Then, $x$ cannot be to the right of $v$ by $M$ being a module, and $x$ cannot be to the left of $v$ since all vertices between $a$ and $v$ are adjacent with $a$. Hence, $uw \in E$, and $M$ is a clique-module. Furthermore, by the choice of $v$, $M$ is a maximal clique-module. It remains to show that $N_G(M)$ is a minimal separator of $G$.

Let $b$ be the rightmost neighbour of $a$ with respect to $\sigma$. Since $G$ is not complete and connected, $b \in N_G(M)$. Let $x$ be the leftmost vertex for which there is an $x, z$-path $P$ in $G$ containing no neighbour of $a$ except for $x$ itself. Clearly, $v \prec_\sigma x \prec_\sigma b$ or $x = b$. By Lemma 11.6, we assume $P$ to be right-monotone. We show that for every vertex $u$, $x \prec_\sigma u \prec_\sigma b$, there is a right-monotone $u, z$-path that does not contain a neighbour of $a$ except for $u$. Suppose there is a vertex $u$, $x \prec_\sigma u \prec_\sigma b$, for which there is no such $u, z$-path; let $u$ be rightmost with this property. Let $c, d$ be adjacent vertices on $P$ such that $c \prec_\sigma u \prec_\sigma d$. If $ud \in E$, we have an appropriate $u, z$-path; if $ud \notin E$, there is a neighbour $w$ of $u$ to the right of $d$ that is not a neighbour of $a$. If $w$ is to the right of $b$, we have found a suitable path, if $w$ is to the left of $b$, there is a $w, z$-path not containing a neighbour of $a$. If there is no neighbour of $a$ between $v$ and $x$, all vertices in $N_G(M)$ are between $x$ and $b$ ($x$ and $b$ included). Then, $G \setminus N_G(M)$ contains two $N_G(M)$-full components (of $G$), namely $G[M]$ and the component containing $z$. Hence, $N_G(M)$ is a minimal separator of $G$.

Finally, we show that all vertices to the left of $x$ are contained in the same module. Suppose there is a vertex $u$ to the left of $x$ that has a neighbour to the right of $x$ or $x$ itself that is not a neighbour of $a$. Then, there is a $u, z$-path containing no neighbour of $a$ except for $u$, contradicting the choice of $x$. Suppose there is a neighbour $w$ of $a$, $u \prec_\sigma w$, that is not a neighbour of $u$. Then, $u$ has a neighbour to the right of $w$ that is not a neighbour of $a$, which contradicts the statement before. Therefore, $M' =_{\text{def}} \{u : u \prec_\sigma x\}$ is a module. Let $\sigma'$ denote the restriction of $\sigma$ to the vertices of $M'$. Due to Lemma 11.6, $\sigma'$ is a min-LexBFS-ordering for $G[M']$, and by induction hypothesis, $N_G(M) \cap M'$ is a minimal separator of $G[M']$. (If $M'$ is not a connected module, we can easily restrict to the connected component of $G[M']$ containing the vertices of $M$, which is a clique in $G$.) Applying Lemma 11.7, $N_G(M)$ is a minimal separator of $G$, and we conclude the theorem.

■

We know now that LexBFS and min-LexBFS identify moplexes. Is it true that both algorithms find the same moplexes? This question also relates to the question whether our result for min-LexBFS is an "easy" corollary of the result for LexBFS. To answer both questions, consider the graph depicted in Figure 26. This graph contains exactly five moplexes labelled with the letters $a, b, c, d, e$—all moplexes of the graph have size 1. Independent of the start vertex, LexBFS can generate orderings with only $b$ or $d$ as first vertex, whereas min-LexBFS numbers only $a$ or $c$ last. Moplex $\{e\}$ can be generated by neither of both algorithms.

## 11.3 Getting to know AT-free claw-free graphs

The claw, or $K_{1,3}$, is the graph on four vertices that contains three edges and three pairwise independent vertices. In other words, the claw is the graph that is built from an independent set of size 3 and an additional universal vertex. A copy is depicted in Figure 2.

**Definition 49** *A graph is **claw-free** if and only if it does not contain the claw as induced subgraph.*

Thus, similar to $2K_2$-free graphs, claw-free graphs are defined by a forbidden induced subgraph on four vertices. However, claw-freeness does not guarantee comparable robust structural properties as $2K_2$-freeness. As an example, the class of claw-free graphs is not closed under minimal fill-in. To see this, consider the chordless cycle on six vertices, $C_6$. This graph is claw-free. Making one of its vertices universal, defines a minimal triangulation of $C_6$, and this triangulation contains a claw. It is easily seen that line graphs of arbitrary graphs are claw-free: there cannot be three edges in a graph that are mutually non-adjacent and are adjacent with a common neighbouring edge.

We restrict our considerations to a subclass of the class of claw-free graph. As we will see, this class provides enough structure to ensure robust properties.

**Definition 50** *A graph is **AT-free claw-free** if and only if it is AT-free and claw-free.*

Hence, the class of AT-free claw-free graphs is just the intersection of the classes of AT-free and claw-free graphs. These graphs and their properties have been studied by several authors in the past [69], [50], [40]. A structural characterisation of AT-free claw-free graphs was given by Kloks, Kratsch, Müller [49]. A graph is *triangle-free* if

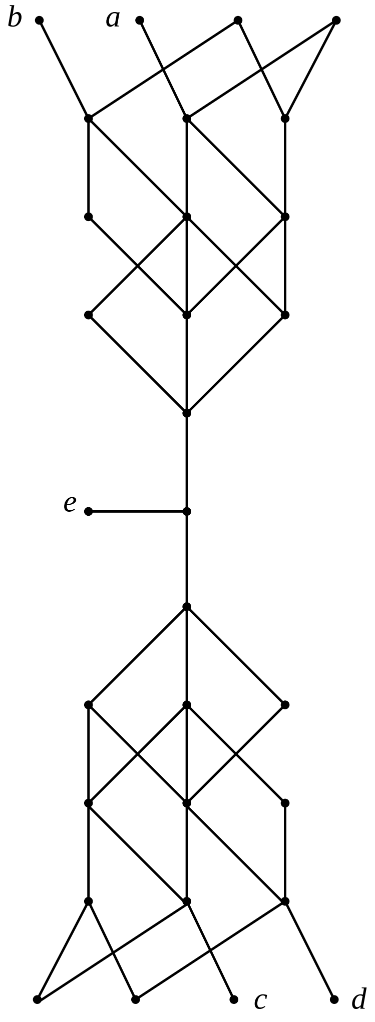


**Figure 26** Is it true that `LexBFS` and `min-LexBFS` always find the same moplexes?—No!, it is not.

it does not contain the triangle as induced subgraph, i.e., if it does not contain three pairwise adjacent vertices.

**Theorem 11.9 (Kloks, Kratsch and Müller, [49])**
*A connected graph is AT-free claw-free if and only if it is a claw-free cocomparability graph or the complement of a triangle-free graph.*

Another interesting structural property of AT-free claw-free graphs has been given by Hempel and Kratsch [40]. We state their result in a way that is more general. Vertex $z$ of the following lemma has been required by Hempel and Kratsch to be the first of a LexBFS-ordering, which is a moplex vertex due to results by Berry and Bordat [5].

**Lemma 11.10 (Hempel and Kratsch, [40])**
*Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $z$ be a moplex vertex of $G$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z$. Then, the following holds:*

*(1) $S_0, S_2, \ldots, S_k$ are cliques in $G$*

*(2) for every pair $u, v$ of non-adjacent vertices from $S_1$, $u$ and $v$ have neighbours in $S_2$.*

**Proof:** Let $M \subseteq V$ be the moplex of $G$ containing $z$. It holds that $M \subseteq S_0 \cup S_1$. If there are two non-adjacent vertices $u$ and $v$ in $S_1$, neither of them belongs to $M$, i.e., they are contained in $N_G(M)$. Since $N_G(M)$ is a minimal separator of $G$, $u$ and $v$ both have neighbours outside $N_G[M]$, hence in $S_2$. This proves statement (2).

For proving statement (1), let $G' =_{\text{def}} G \setminus N_G(M)$. Let $u$ and $v$ be vertices in $S_2$. If $u$ and $v$ have a common neighbour $w$ in $S_1$, then $\{u, v, w, z\}$ induces a claw in $G$ if and only if $uv \notin E$. Since $G$ is claw-free, $uv \in E$. Let $u$ and $v$ do not have a common neighbour in $S_1$. If they are contained in the same connected component of $G'$, there is a $u,v$-path in $G$ avoiding the neighbourhood of $z$, so that $u, v, z$ forms an asteroidal triple in $G$ if and only if $uv \notin E$. If $u$ and $v$ are not contained in the same connected component of $G'$, some neighbour in $S_1$ of $u$ or of $v$ must have neighbours in $S_2$ that are in two connected components of $G'$, and we have a claw in $G$, so that this case is not possible. Remember that $N_G(M)$ is a minimal separator of $G$, and at least one of the connected components of $G'$ is $N_G(M)$-full in $G$. Hence, $S_2$ is a clique in $G$. Now, let $u$ and $v$ be vertices in $S_{i+1}$ for $i \in \{2, \ldots, k-1\}$. Again, if $u$ and $v$ have a common neighbour in $S_i$, they must be adjacent on $G$. If $u$ and $v$ do not have a common neighbour in $S_i$, $u, v, z$ form an asteroidal triple in $G$ if and only if $uv \notin E$. Hence, $S_{i+1}$ is a clique.

■

Based on the result of Lemma 11.10, Hempel and Kratsch showed that the diameter of an AT-free claw-free graph can be determined using moplex vertices.

**Corollary 11.11 (Hempel and Kratsch, [40])**
*Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $z$ be a moplex vertex of $G$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z$. Then,* $\operatorname{diam}(G) = k$.

**Proof:** If $G$ is complete, $k \leq 1$ and $\operatorname{diam}(G) = k$. So, let $G$ be non-complete. Hence, $k \geq 2$. (This is due to the moplex property of $z$, i.e., there is a vertex of $G$ that is non-adjacent with $z$.) Let $u$ and $v$ be vertices of $G$. If they are in the same BFS-level, they are adjacent or have a common neighbour due to Lemma 11.10. Otherwise, let $i, j \leq k$ such that $i < j$ and $u \in S_i$ and $v \in S_j$. Let $x \in S_i$ such that there is an $x, v$-path in $G$ of length $j - i$. If $i \geq 2$, there is a $u, v$-path of length at most $j - i + 1 \leq k - 1$ in $G$ due to Lemma 11.10. If $u = z$, $u = x$ and there is a $u, v$-path of length at most $k$ in $G$. It remains the case $i = 1$. Let $M$ be the moplex of $G$ containing $z$. If $u \in M$, $u$ and $x$ are adjacent, and there is a $u, v$-path of length at most $k$. If $u \in N_G(M)$, $u$ has a neighbour $w$ in $S_2$, and the $x, v$-path contains a vertex from $S_2$. We can construct a $u, v$-path of length at most $k$. Hence, $G$ does not contain shortest paths of length more than $k$.

■

For the study of minimal triangulations of AT-free claw-free graphs, we need a subclass of chordal graphs, even of interval graphs.

**Definition 51** *A graph is a* ***proper interval graph*** *if and only if it is an interval graph that has an interval model such that no interval is properly contained in another.*

We will see later that proper interval graphs exactly correspond to AT-free claw-free graphs in the sense of the relationship between interval graphs and AT-free graphs (Theorem 10.22). We give a first characterisation of proper interval graphs.

**Theorem 11.12 (Roberts, [73])**
*A graph is a proper interval graph if and only if it is chordal AT-free claw-free.*

Similar to interval graphs, proper interval graphs can be characterised by special vertex orderings.

**Definition 52** *Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. We say that $\sigma$ is a* ***proper interval ordering*** *for $G$ if and only if for every triple $u, v, w$ of vertices of $G$ holds:*

$$u \prec_\sigma v \prec_\sigma w \text{ and } uw \in E \quad \Longrightarrow \quad uv \in E \text{ and } vw \in E\,.$$

In other words, a vertex ordering $\sigma$ for a graph $G$ is a proper interval ordering for $G$ if and only if it and its reverse, which means that $\sigma$ is read from right to left, are interval orderings for $G$. It also follows that $\sigma$ is a proper interval ordering for $G$ if and only if its reverse is a proper interval ordering for $G$.

**Theorem 11.13 (Looges and Olariu, [58])**
*A graph is a proper interval graph if and only if it has a proper interval ordering.*

**Proof:** Let $G = (V, E)$ be a proper interval graph, and let $\mathfrak{I}$ be an interval ordering for $G$ such that no interval is properly contained in another. Note that two intervals may be equal. Let $\tau$ be a vertex ordering for $G$. Let $I_x = [\ell(x), r(x)]$ be the interval for $x \in V$. Let $\sigma$ be a vertex ordering for $G$ such that, for every pair $u, v$ of vertices of $G$, $u \prec_\sigma v$ if and only if $r(u) < r(v)$ or, if $r(u) = r(v)$, $\ell(u) < \ell(v)$ or, if $r(u) = r(v)$ and $\ell(u) = \ell(v)$, $u \prec_\tau v$. We show that $\sigma$ is a proper interval ordering for $G$. So, let $u, v, w$ be vertices of $G$ such that $u \prec_\sigma v \prec_\sigma w$ and $uw \in E$. Then, $r(u) \leq r(v) \leq r(w)$ by the definition of $\sigma$ and $\ell(w) \leq r(u)$. Then, $I_v$ and $I_w$ have a non-empty intersection, i.e., $vw \in E$. Since $\mathfrak{I}$ does not contain an interval that is properly contained in another, $\ell(v) \leq \ell(w) \leq r(u)$, and $uv \in E$.

For the converse, let $G = (V, E)$ be a graph on $n$ vertices, and let $\sigma$ be a proper interval ordering for $G$. We construct an appropriate interval model for $G$ from $\sigma$. For every vertex $x$ of $G$, let $a_x$ denote the smallest number $i \in \{1, \ldots, n\}$ such that $\sigma(i) \in N_G[x]$. Similarly, let $b_x$ denote the largest number $i \in \{1, \ldots, n\}$ such that $\sigma(i) \in N_G[x]$. Then, for every vertex $x$ of $G$, let $\ell(x) =_{\text{def}} a_x \cdot n + \sigma^{-1}(x)$ and $r(x) =_{\text{def}} b_x \cdot n + \sigma^{-1}(x)$. This construction has already been used in the proof of Theorem 10.5, so that we know that $\mathfrak{I} =_{\text{def}} \{I_x =_{\text{def}} [\ell(x), r(x)]\}_{x \in V}$ defines an interval model for $G$. It remains to show that no interval of $\mathfrak{I}$ is properly contained in another. Suppose there are vertices $u$ and $v$ of $G$ such that $\ell(v) \leq \ell(u)$ and $r(u) \leq r(v)$ and one of these relations is strict. If $u \prec_\sigma v$, i.e., $\sigma^{-1}(u) < \sigma^{-1}(v)$, $\ell(v) \leq \ell(u)$ means $\ell(v) < \ell(u)$ and $N_G[v]$ contains a vertex that is farther to the left with respect to $\sigma$ than any vertex in $N_G[u]$ ($a_v < a_u$). This, however, is not possible, since $\sigma$ is a proper interval ordering. If $v \prec_\sigma u$, we similarly conclude that $r(u) \leq r(v)$ means that $N_G[v]$ contains a vertex that is farther to the right than any vertex in $N_G[u]$, which is not possible. Hence, we conclude the theorem.

■

Observe that the proof of Theorem 11.13 provides a linear-time algorithm, given a graph $G$ and a proper interval ordering for $G$, to compute an interval model for $G$ that does not contain an interval that is properly contained in another.

## 11.4 Characterisations of minimal triangulations of AT-free claw-free graphs

We will present two characterisations of minimal triangulations of AT-free claw-free graphs. The first one is taken from the literature and is the AT-free claw-free analogue to Theorems 9.6 and 10.22. The second characterisation can be considered the AT-free claw-free counterpart of Theorems 9.10 and 10.27. However, the characterising means is completely different: It involves min-LexBFS-orderings.

**Theorem 11.14 (Parra and Scheffler, [69])**
*A graph is AT-free claw-free if and only if all its minimal triangulations are AT-free claw-free.*

**Proof:** Let $G = (V, E)$ be a graph. Due to Theorem 10.22, $G$ is AT-free if and only if all minimal triangulations of $G$ are AT-free, hence interval graphs. It remains to show that, if $G$ is AT-free, $G$ is claw-free if and only if all minimal triangulations of $G$ are claw-free. Let $G$ be not claw-free, and let $\{u_1, u_2, u_3, v\}$ induce a claw. Then, $\{u_1, u_2, u_3, v\}$ induces a claw in $H' =_{\text{def}} G \cup F'$ where

$$F' =_{\text{def}} \{xz \notin E : x \neq z \text{ and } |\{x, z\} \cap \{u_1, u_2, u_3, v\}| \leq 1\}.$$

$H'$ is chordal, even an interval graph. Since $\{u_1, u_2, u_3, v\}$ induces a claw in every graph $G'$ that is between $G$ and $H'$, i.e., $G \subseteq G' \subseteq H'$, there is a minimal triangulation $H \subseteq H'$ of $G$ that contains a claw. Now, let $H$ be a minimal triangulation of $G$ that contains a claw. Since $G$ is AT-free, $H$ is an interval graphs. Let $\langle A_1, \ldots, A_k\rangle$ be a consecutive clique arrangement for $H$. Let $\{u_1, u_2, u_3, v\}$ induce a claw in $H$, and let $v$ be the vertex adjacent with $u_1, u_2, u_3$ in $H$. Let $r \in \{1, \ldots, k\}$ be smallest such that $v \in A_r$, and let $s \in \{1, \ldots, k\}$ be largest such that $v \in A_s$. Since $\{u_1, u_2, u_3\}$ is an independent set in $H$, $u_1, u_2, u_3$ appear only in different maximal cliques of $H$. Hence, $r + 1 < s$. Let $w_1 \in A_r \setminus A_{r+1}$ and $w_3 \in A_s \setminus A_{s-1}$. Note that $w_1$ and $w_3$ are adjacent with $v$ in $G$. Let $M =_{\text{def}} (A_{r+1} \cup \cdots \cup A_{s-1}) \setminus (A_r \cup A_s)$. If there is a vertex $w_2 \in M$ that is adjacent with $v$ in $G$ then $\{w_1, w_2, w_3, v\}$ induces a claw in $G$. We will show in the paragraph below that another case is not possible.

We assume that $v$ is adjacent with no vertex from $M$. Consider graph $H_1 =_{\text{def}} H \setminus A_r$. Let $A'_{r+1} =_{\text{def}} A_{r+1} \setminus A_r$ and $A'_s =_{\text{def}} A_s \setminus A_r$. Suppose $A'_{r+1} \cap A'_s = \emptyset$, and the vertices of $A'_{r+1}$ and $A'_s$ are in different connected components of $H_1$; let $D$ be the connected component of $H_1$ containing the vertices of $A'_{r+1}$. Then, $H \setminus \{vx : x \in V(D)\}$ is a proper subgraph of $G$ and chordal, i.e., a triangulation of $G$, which is not possible due to minimality of $H$. We conclude that the vertices of $A'_{r+1}$ and $A'_s$ are not in different connected components of $H_1$. Similarly, it follows that the vertices in $A'_r =_{\text{def}} A_r \setminus A_s$ and $A'_{s-1} =_{\text{def}} A_{s-1} \setminus A_s$ are not in different connected components of $H_2 =_{\text{def}} H \setminus A_s$. Then, $w_1, w_2, w_3$, where $w_2 \in M$, form an asteroidal triple in $G$, which is not possible by assumption.

■

We can draw the following result from the proof of Theorem 11.14: Let $G$ be a graph, and let $H$ be a minimal triangulation of $G$ that is an interval graph. If $H$ contains a claw then $G$ contains a claw or an asteroidal triple. Again, we see that (and why) the class of claw-free graphs is not closed under minimal fill-in.

We will prove our algorithmic characterisation of minimal triangulations of AT-free claw-free graphs in two steps. First, we will show that 2free-min-LexBFS-orderings for AT-free claw-free graphs represent minimal triangulations in a special way. To precise this idea, we make the following definition. Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. By $H = (G, \sigma)$, we denote the *filled* graph of $G$ with respect to $\sigma$, and $H$ is the result of the following completion process. If $G$ contains no vertex, then $H =_{\text{def}} G$. If $u$ is the leftmost vertex of $G$ with respect to $\sigma$, let $G'$ be the graph obtained from $G$ by making into a clique the neighbourhood of $u$ and removing $u$. Let $\sigma'$ be obtained from $\sigma$ by removing $u$. Then, $H =_{\text{def}} (G', \sigma') + u$ where $u$ has the same neighbourhood in $H$ as in $G$. In other words, $H$ is obtained from $G$ by making vertices simplicial and processing them according to $\sigma$. Note that $\sigma$ is a perfect elimination scheme for $H$, hence $H$ is chordal, and $H$ is a triangulation of $G$. If $H$ is a minimal triangulation of $G$, we call $\sigma$ a *minimal elimination scheme* for $G$. If $\sigma$ is a minimal elimination scheme for $G$ and an interval ordering for $(G, \sigma)$, we call it a *minimal interval elimination scheme* for $G$.

**Lemma 11.15** *Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $\sigma$ be a 2free-min-LexBFS-ordering for $G$. Then, $\sigma$ is a minimal interval elimination scheme for $G$.*

**Proof:** Let $H =_{\text{def}} (G, \sigma)$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z = \sigma(n)$. We will show that $\sigma$ is a minimal interval elimination scheme for $G$. This proof is accomplished in three steps: first, we show that the BFS-levels of $G$ are also the BFS-levels of $H$, second, we show that $\sigma$ is an interval ordering for $H$, and third, we show that $\sigma$ is a minimal elimination scheme for $G$.

(1) Let $S'_0, \ldots, S'_\ell$ be the BFS-levels of $H$ with root $z$. Remember that $\sigma$ is a BFS-ordering for $G$ due to Lemma 11.2. Suppose there is $i$ such that $S_i \neq S'_i$; let $i$ be smallest possible. It is clear that $S_0 = S'_0$, and due to Lemma 11.10, $S_1 = S'_1$. Hence, $i \geq 2$. Since $S_{i-1} = S'_{i-1}$, $S_i \subseteq S'_i$, and there is a vertex $u \in S'_i \setminus S_i$. By definition of $H$, there must be vertices $v \in S_{i-1}$ and $x$, $x \prec_\sigma u$, such that $x$ is a common neighbour of $u$ and $v$ in $G$. Then, $x \in S_i$, and by $x \prec_\sigma u$, $u \in S_i$, too, which contradicts the assumption.

(2) Remember that $z$ is a vertex in a moplex $M$ of $G$. According to Lemma 11.10, $S_2$ is a clique in $G$. Let $u \in S_2$ be rightmost with respect to $\sigma$. During the fill process of $(G, \sigma)$, $u$ becomes adjacent with every vertex in $N_G(M)$. So, when $u$ is processed, $N_G(M)$ becomes a clique. Hence, $S_1$ is a clique in $H$, i.e., every BFS-level is a clique in $H$ due to Lemma 11.10 and $z$ being a moplex vertex. Suppose there are vertices $u, v, w \in V$, $u \prec_\sigma v \prec_\sigma w$, such that $uw \in E(H)$ and $vw \notin E(H)$. Then, $u, v \in S_i$ and $w \in S_{i-1}$ for some $i \geq 2$, and $uv \in E$. According to the filling process defined by $(G, \sigma)$ and $uw \in E(H)$, $u$ and $w$ are adjacent when $u$ is chosen to be made simplicial. This completion, however, adds edge $vw$. So, $\sigma$ is an interval ordering for $H$.

(3) To show that $\sigma$ is a minimal elimination scheme for $G$, it suffices to show for every edge in $F =_{\text{def}} E(H) \setminus E$ that it is unique chord in a cycle of length 4 in $H$ due to Theorem 8.14. Let $uv \in F$, $u \prec_\sigma v$, $u \in S_j$, $v \in S_i$, $i \leq j$. It holds

that $\mathrm{lm}_G(v) \prec_\sigma u$ by the definition of $(G, \sigma)$; otherwise, no vertex can introduce edge $uv$ to $H$. If $i = j$, then $i = j = 1$, since all other BFS-levels are cliques in $G$ due to Lemma 11.10. If $u$ and $v$ have a common neighbour $x \in S_2$, $x$ and $z$ are not adjacent in $H$ (remember that $G$ and $H$ have the same BFS-levels with root $z$), and $uv$ is unique chord in the cycle $(u, x, v, z)$. Otherwise, let $w$ and $x$ from $S_2$ be neighbours of $u$ and $v$, respectively. Since $H$ is chordal, $ux \in F$ or $vw \in F$ must hold. Then, $uv$ is unique chord in the cycle $(u, x, v, z)$ or $(u, w, v, z)$. Let $1 \leq i < j$. Then, $j = i + 1$, and $\mathrm{lm}_G(v) \in S_j$ and $\mathrm{lm}_G(v)$ is a common neighbour of $u$ and $v$ in $G$ by Lemma 11.10. Since $\sigma$ has the farther right neighbour property, $u$ has a neighbour $w$ to the right of $v$ with respect to $\sigma$ that is not adjacent with $\mathrm{lm}_G(v)$ in $H$ (due to $\mathrm{lm}_G(v) \prec_\sigma \mathrm{lm}_G(w)$, no vertex to the left of $\mathrm{lm}_G(v)$ can introduce an edge between $\mathrm{lm}_G(v)$ and $w$), and $v$ and $w$ are contained in the same BFS-level. Then, $uv$ is unique chord in the cycle $(\mathrm{lm}_G(v), u, w, v)$.

■

In the proof of Lemma 11.15, we have shown that, for $G$ an AT-free claw-free graph and $\sigma$ a 2free-min-LexBFS-ordering, the BFS-levels of $G$ are also the BFS-levels of $(G, \sigma)$. This property, in fact, holds for arbitrary graphs and BFS-orderings, since no special property has been used. We want to show next that every minimal triangulation of an AT-free claw-free graph can be obtained in the way proposed in Lemma 11.15. To achieve this, we have to show another result about BFS-levels of AT-free claw-free graphs.

**Lemma 11.16** *Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $H$ be a minimal triangulation of $G$. Let $z$ be a moplex vertex of $H$, and let $S_0, \ldots, S_k$ be the BFS-levels of $H$ with root vertex $z$. Then, $S_0, \ldots, S_k$ are the BFS-levels of $G$ with root vertex $z$.*

**Proof:** Let $S'_0, \ldots, S'_\ell$ be the BFS-levels of $G$ with root vertex $z$. Obviously, $S_0 = S'_0$. Since $G$ is a spanning subgraph of $H$, $\ell \geq k$. Suppose there is $i \in \{1, \ldots, k\}$ such that $S_i \neq S'_i$; let $i$ be smallest possible. It holds that $S'_i \subset S_i$; let $u \in S_i \setminus S'_i$, which means that $u$ does not have a neighbour in $S_{i-1} = S'_{i-1}$ in $G$. If $i = 1$, $uz$ is unique chord in a cycle of length 4 in $H$ due to Theorem 8.14. Since $z$ is a moplex vertex, $S_1$ is a clique in $H$, and $uz$ cannot be unique chord in such a cycle. Hence, $i \geq 2$. We delete all edges between $u$ and vertices from $S_{i-1}$ in $H$ and obtain graph $H'$. Suppose there is a chordless cycle $C$ of length at least 4 containing $u$. It is easy to see that $C$ can contain at most two vertices from $S_i$, since $H$ is an AT-free claw-free graph due to Theorem 11.14 and $S_i$ is a clique due to Lemma 11.10. But then $C$ is a chordless cycle in $H'[S_i \cup \cdots \cup S_k] = H[S_i \cup \cdots \cup S_k]$ which contradicts $H$ being chordal.

■

From this lemma, we can derive an interesting corollary: For $G$ an AT-free claw-free graph, it holds that the minimal triangulations of $G$ have the same diameter as $G$. Since proper interval graphs are AT-free claw-free graphs due to Theorem 11.12, BFS-levels with root vertex a moplex vertex give the diameter due to Corollary 11.11. This number is equal to the diameter of $G$.

The claim of Lemma 11.16 can be stated in another way. It is said that, if $G$ is a connected AT-free claw-free graph and $H$ is a minimal triangulation of $G$, every BFS-ordering for $H$ that ends with a moplex vertex is a BFS-ordering for $G$. A similar, but weaker, correspondence holds for min-LexBFS-orderings. Since this correspondence involves 2free-min-LexBFS-orderings, we have to know what properties these orderings have.

**Lemma 11.17** *Let $G = (V, E)$ be a connected graph, and let $\sigma$ be a 2free-min-LexBFS-ordering for $G$. If $\sigma$ is not an interval ordering for $G$, then $G$ contains an asteroidal triple, a chordless cycle of length at least 4 or a claw.*

**Proof:** Let $z =_{\text{def}} \sigma(n)$, and let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z$. If $\sigma$ is not an interval ordering there must be a rightmost vertex $w$ such that there is a rightmost vertex $v$ for which holds $\text{lm}(w) \prec_\sigma v \prec_\sigma w$ and $vw \notin E$. Let $u =_{\text{def}} \text{lm}(w)$. Note that $z \neq w$. Let $v \in S_j$ and $w \in S_i$. Since $uw \in E$, $1 \leq i \leq j \leq i{+}1$. Let $i = j = 1$. Since $z$ is a moplex vertex, $v$ and $w$ are not contained in the moplex defined by $z$, so they have neighbours in $S_2$. If they have a common neighbour $a$, $\{a, v, w, z\}$ induces a chordless cycle of length 4 in $G$. If they do not have a common neighbour in $S_2$, $v$ and $w$ have neighbours in one connected component of $G' =_{\text{def}} G \setminus N[z]$, and there is a shortest $v, w$-path $P$ in $G'{+}v, w$. We obtain a chordless cycle $P + z$ of length at least 5. Let $i = j \geq 2$. Let $x \in S_{i-1}$ be a neighbour of $v$. By the choice of $v$ and $w$, $x$ is a neighbour of $w$ and has itself a neighbour $a \in S_{i-2}$, so that $\{v, w, x, a\}$ induces a claw in $G$. Let $1 \leq i < j$. This particularly means that $u \in S_j$. By the farther right neighbour property of $\sigma$, there is a neighbour $a \in S_i$ of $v$ to the right of $w$ that is not a neighbour of $u$. By the choice of $v$ and $w$, $a$ is a common neighbour of $v$ and $w$. If $uv \in E$, $\{u, v, w, a\}$ induces a chordless cycle of length 4. Let $uv \notin E$. Then, $\{u, w, a, v\}$ induces a chordless path of length 3. If $j \geq 3$, then $u$, $v$, $z$ form an asteroidal triple. Finally, let $j = 2$, i.e., $a, w \in S_1$ and $v, u \in S_2$. If $u$ and $v$ are contained in the same connected component of $G \setminus S_1$, then $u$, $v$, $z$ form an asteroidal triple. If they are not contained in one connected component, $a$ or $w$ (being neighbours of $z$) must have neighbours in two different connected components of $G \setminus S_1$ (besides the connected component containing $z$), since $S_1$ is a minimal separator. Then, $a$ or $w$ has two non-adjacent neighbours in $S_2$, and we find a claw in $G$.

■

**Lemma 11.18** *Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $H$ be a minimal triangulation of $G$. Let $\sigma$ be a 2free-min-LexBFS-ordering for $H$. Then, $\sigma$ is a 1free-min-LexBFS-ordering for $G$.*

**Proof:** Since $H$ is a proper interval graph due to Theorem 11.14, hence chordal AT-free claw-free due to Theorem 11.12, $\sigma$ is an interval ordering for $H$ due to Lemma 11.17. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z = \sigma(n)$; $z$ is a moplex vertex. We try to generate $\sigma$ by a `min-LexBFS` run on $G$, i.e., we try to always choose the vertex determined by $\sigma$. In other words, we want to show that $\sigma = \texttt{min-LexBFS}^*(G, \vec{\sigma})$, where $\vec{\sigma}$ is the reverse of $\sigma$. Suppose that this is not possible. Then, there is a rightmost vertex $x$ that, during the generation process by `min-LexBFS`$^*$, separates vertices $v$ and $u$ where $u \prec_\sigma v$. Without loss of generality, we assume $u = \mathrm{lm}_G(x)$. Since $\sigma$ is a BFS-ordering for $G$ due to Lemma 11.16, $u$ and $v$ must be contained in the same level $S_i$ for some $i \geq 1$. If $i = 1$, then $x, u, v$ are vertices in $N_G(z)$ ($x \neq z$, since $vx \notin E$), and $x$ has a neighbour in $S_2$ by the moplex property of $z$, which contradicts $u = \mathrm{lm}_G(x) \in S_1$. So, $i \geq 2$, $x \in S_{i-1}$ and $u$ and $v$ are adjacent due to Lemma 11.10. By the definition of interval orderings, every neighbour of $u$ to the right of $v$ with respect to $\sigma$ is a neighbour also of $v$, and so, $vx \in E(H) \setminus E$. Since $H$ is a minimal triangulation of $G$, there are non-adjacent vertices $a$ and $b$, $a \prec_\sigma b$, such that $vx$ is unique chord in the cycle $(a, v, b, x)$ in $H$. Then, $a \prec_\sigma v \prec_\sigma x \prec_\sigma b$, and $u$ is not to the right of $a$. Since $u$ and $b$ are not adjacent in $H$ and since $\mathrm{lm}_H(b) = \mathrm{lm}_G(b)$ is not to the right of $v$, there is a vertex to the right of $x$ that separates $u$ and $v$ (it may be $b$ or is a vertex to the right of $b$ according to the farther right neighbour property of $\sigma$), which contradicts the assumption. Hence, $\sigma$ is a 1free-min-LexBFS-ordering for $G$.

■

**Lemma 11.19** *Let $G = (V, E)$ be a connected AT-free claw-free graph. Let $H$ be a minimal triangulation of $G$, and let $\sigma$ be an interval ordering for $H$. Then, $H = (G, \sigma)$.*

**Proof:** Let $H' =_{\mathrm{def}} (G, \sigma)$. Remember that $H'$ is chordal. Suppose $H' \not\subseteq H$. Since $H$ is a minimal triangulation of $G$, there must be an edge in $H'$ that is not in $H$. Let $u$ be the leftmost vertex with regard to $\sigma$ that causes an edge between two vertices $v$ and $w$ during the fill process that is not an edge in $H$. By the choice of $u$, it holds that $uv$ and $uw$ are edges in $H$. Since $u$ is leftmost, we can assume that $u \prec_\sigma v \prec_\sigma w$. But since $\sigma$ is an interval ordering for $H$, $vw$ is an edge in $H$, which contradicts the assumption. So, $H' \subseteq H$, and thus $H' = H$.

■

Now, it is easy to characterise the minimal triangulations of an AT-free claw-free graph.

**Theorem 11.20** *Let $G$ be a connected AT-free claw-free graph, and let $H$ be a graph. Then, $H$ is a minimal triangulation of $G$ if and only if there is a 2free-min-LexBFS-ordering $\sigma$ for $G$ such that $H = (G, \sigma)$.*

**Proof:** If $\sigma$ is a 2free-min-LexBFS-ordering for $G$, $\sigma$ is a minimal interval elimination scheme for $G$ due to Lemma 11.15. In particular, $(G, \sigma)$ is a minimal triangulation of $G$. For the converse, let $H$ be a minimal triangulation of $G$. Due to Theorem 11.14, $H$ is a proper interval graph, and due to Theorem 11.13, there is a proper interval ordering $\sigma$ for $H$. It holds that $\sigma$ and $\vec{\sigma}$, the reverse of $\sigma$, have the farther right neighbour property, and $\sigma = \texttt{min-LexBFS}^*(H, \vec{\sigma})$ and $\vec{\sigma} = \texttt{min-LexBFS}^*(H, \sigma)$. Hence, $\sigma$ and $\vec{\sigma}$ are 2free-min-LexBFS-orderings for $H$ and 1free-min-LexBFS-orderings for $G$ due to Lemma 11.18. Furthermore, $\sigma = \texttt{min-LexBFS}^*(G, \vec{\sigma})$. Therefore, $\sigma$ is a 2free-min-LexBFS-ordering for $G$. Since $\sigma$ is an interval ordering for $H$, $H = (G, \sigma)$ due to Lemma 11.19.

■

## 11.5 Certifying recognition of proper interval graphs

For solving the min-Tri membership problem for AT-free claw-free graphs, we need an algorithm that recognises proper interval graphs and additionally generates a proper interval ordering. In the literature, there are a number of proper interval graph recognition algorithms, such as in [20], [24], [41], [67]. All these algorithms apply a breadth first search strategy more or less directly, and they generate appropriate representations. However, these algorithms are rather involved. In contrast, we will present an easy algorithm that computes a proper interval ordering for the input graph, if it is a proper interval graph. Our algorithm is based on `min-LexBFS`. Independently, Corneil developed a similar algorithm that is based on `LexBFS` [19].

The proper interval graph recognition algorithm in this section has an additional property with respect to all algorithms cited from the literature. All recognition algorithms above provide a so-called "certificate" for the case the input graph is a proper interval graph. However, if the input graph is not a proper interval graph, they just say "no", and the user has to trust the algorithm. Our algorithm tries to *convince* the user of its decision, in the accepting as well as in the rejecting case. More precisely, if the input graph is a proper interval graph, the algorithm outputs a proper interval ordering for the graph, if the input graph is not a proper interval graph, the algorithm outputs a chordless cycle of length at least 4, an asteroidal triple or a claw. This approach is motivated by Theorems 11.12 and 11.13.

Certifying algorithms, as this class of algorithms is called, are highly interesting in practice, especially if their proofs of correctness or their implementations are rather complicated or if the user requires a high level of security [86], [62]. There are only few such algorithms known, among the first ones is the well-known recognition algorithm for bipartite graphs. It outputs a 2-colouring of the input graph or a cycle of odd length. It can easily be verified whether the certificate proves the result. Recently, Kratsch, McConnel, Mehlhorn, Spinrad presented certifying algorithms for the recognition of interval graphs and permutation graphs [53]. Their interval graph recognition algorithm is based on Theorem 10.4 and outputs an interval model, if the input graph is an interval graph, or an asteroidal triple or a chordless cycle of length greater than 3, if the input graph is not an interval graph. Later, Hell and Huang presented another certifying recognition algorithm for proper interval graphs, that is based on `LexBFS` [39].

Our recognition algorithm is simple: `3min-LexBFS` generates a proper interval ordering for the input graph if and only if it is a proper interval graph. If it is not a proper interval graph, the algorithm outputs, instead of a vertex ordering, certificates proving the result. The 3-sweep breadth first search approach has also been used by Corneil for his proper interval graph recognition algorithm [19]. A similar approach, i.e., multi-sweep breadth first search, was used by Corneil, Olariu, Stewart for recognising interval graphs [22]. Recently, Bretscher, Corneil, Habib, Paul used a multi-sweep breadth first search approach for the recognition of cographs, the graphs without induced $P_4$ [15]. Interestingly, their algorithm is also certifying.

We give a short sketch of our algorithm. It works in two phases: construction and verification. The construction phase is a `3min-LexBFS` where the first sweep finds a moplex vertex of the input graph, the second sweep constructs an interval ordering and the third sweep finally constructs a proper interval ordering. The verification phase verifies whether the output of the first phase is an interval ordering and a reversed interval ordering. If no violation has been encountered, the ordering is indeed a proper interval ordering, and the input graph is accepted. Otherwise, the algorithm determines an asteroidal triple, a claw or a chordless cycle of length at least 4 as certificate for non-membership. Before we prove the central lemma, it is necessary to know that the first and last vertices of a 2min-LexBFS-ordering for an interval graph are at maximal distance, which we will prove first.

**Lemma 11.21** *Let $G = (V, E)$ be a connected interval graph, and let $\sigma$ be a 2free-min-LexBFS-ordering for $G$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root $z =_{\text{def}} \sigma(n)$. For every $u, v \in S_i$, $i \geq 1$, $u$ and $v$ have a common neighbour in $S_{i-1}$.*

**Proof:** Let $\sigma'$ be the result generated by the first `min-LexBFS` sweep. Let $S'_0, \ldots, S'_\ell$ be the BFS-levels of $G$ with root $s =_{\text{def}} \sigma'(n)$. Remember that $z = \sigma'(1)$. If $\ell \leq 1$ then $s$ is a universal vertex in $G$. If $\ell = 1$, then $s \in S_1$, and every vertex in $S_1$ is adjacent with $z$ and every vertex in $S_2$, if $k > 1$, is adjacent with $s$. So, for the remaining proof, we assume $k \geq \ell \geq 2$. Note that $s \in S_2 \cup \cdots \cup S_k$. We first prove two claims that hold for arbitrary graphs.

(1) We show that there are a connected component $C$ of $G \setminus S_1$ and a vertex $y \in V(C) \cap S_2$ such that for every vertex $v \in S_2 \setminus V(C)$: $N_G(y) \cap S_1 \subseteq N_G(v)$. Let $y \in S_2$ be a vertex on a shortest $s, z$-path, and let $C$ be the connected component of $G \setminus S_1$ that contains $s$ and $y$. Hence, $y \in S'_{\ell-2}$ and $N_G(y) \cap S_1 \subseteq S'_{\ell-1}$. It

is clear that $S_1 \subseteq S'_{\ell-1} \cup S'_\ell$, and it follows that $S_2 \setminus V(C) \subseteq S'_\ell$, since every shortest $v,s$-path for $v \in S_2 \setminus V(C)$ contains a vertex from $S_1$. Suppose there is a vertex $v \in S_2 \setminus V(C)$ that is not adjacent with some vertex $w \in N_G(y) \cap S_1$. By construction, $v \in S'_\ell$, $w \in S'_{\ell-1}$, hence $z \prec_{\sigma'} v \prec_{\sigma'} w$. Due to the farther right neighbour property of $\sigma'$ there is a neighbour $x$ of $v$ to the right of $w$ with respect to $\sigma'$ that is not a neighbour of $z$. Since $w \in S'_{\ell-1}$ and $w \prec_{\sigma'} x$ and $vx \in E$, $x \in S'_{\ell-1}$. By construction ($v \in S_2$), $x \in S_1 \cup S_2 \cup S_3$. Note that $s \in S_\ell$ and that $x$ and $s$ are at distance $\ell - 1$. Since $x \notin V(C)$, otherwise $v \in V(C)$, $x \in S_1$ due to the existence of an $x,s$-path of length $\ell - 1$. But then $x$ must be a neighbour of $z$ which contradicts the assumption.

(2) We show that there is at most one connected component in $G \setminus S_1$ that contains vertices from $S_3$. Suppose there are two connected components in $G \setminus S_1$ that contain vertices from $S_3$. Remember that there is $i \geq 2$ such that $s \in S_i$. By our assumption, there is a vertex $x \in S_3$ that is not in the same connected component of $G \setminus S_1$ as $s$, so that $\text{dist}_G(s,x) \geq i-1+2 = i+1 > i = \text{dist}_G(z,s)$. But then, $z$ could not be contained in $S'_\ell$.

For proving the lemma, let $i \geq 2$. First, let $u, v \in S_i$ be vertices in the same connected component $C$ of $G \setminus S_{i-1}$. Observe that $N_G(C)$ is a minimal separator of $G$ and so is a clique in $G$ (Theorem 8.4). Suppose there are a neighbour $a \in S_{i-1}$ of $u$ that is not a neighbour of $v$ and a neighbour $a' \in S_{i-1}$ of $v$ that is not a neighbour of $u$. If $u$ and $v$ are adjacent then $\{u, v, a, a'\}$ induces a chordless cycle; otherwise $u, v, z$ form an asteroidal triple. Hence, $N_G(u) \cap S_{i-1} \subseteq N_G(v)$ or $N_G(v) \cap S_{i-1} \subseteq N(u)$. Second, let $u, v \in S_i$ be vertices in different connected components of $G \setminus S_{i-1}$; in particular, $uv \notin E$. If $i = 2$, let $C$ be the connected component of $G \setminus S_1$ containing $s$ and $y$. If neither $u$ nor $v$ belong to $C$, every neighbour in $S_1$ of $y$ is a neighbour of $u$ and $v$, hence they have a common neighbour in $S_1$. If, without loss of generality, $u$ belongs to $C$, $u$ and $y$ have a common neighbour $w$ in $S_1$, since they are in the same connected component of $G \setminus S_1$. Hence, $w$ is a common neighbour of $u$ and $v$. If $i \geq 3$, $u$ and $v$ are in the same connected component of $G \setminus S_1$. If they have no common neighbour in $S_{i-1}$, $u, v, z$ form an asteroidal triple.

■

**Corollary 11.22** *Let $G = (V, E)$ be a connected interval graph. Let $\sigma$ be a 2free-min-LexBFS-ordering for $G$, and let $a =_{\text{def}} \sigma(1)$ and $z =_{\text{def}} \sigma(n)$. Then,* $\text{diam}(G) = \text{dist}_G(z, a)$.

**Proof:** Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root $z$. If $k \leq 1$, then $G$ is complete, and the statement holds. Let $k \geq 2$, and let $u \in S_i$ and $v \in S_j$ for $i \leq j$. If $i = 0$, which means that $u = z$, then $\text{dist}_G(u,v) \leq k$. If $i = 1$ then $\text{dist}_G(u,v) \leq k$, since $S_1$ is a minimal separator of $G$ and hence a clique. If $i \geq 2$ then $\text{dist}_G(u,v) \leq j - i + 2 \leq k$ due to Lemma 11.21.

■

The main result of this section is contained in the following lemma. However, one part of it has already been proved—it is Lemma 11.17. The combination of the proofs of both lemmata describes an algorithm that finds an asteroidal triple, a chordless cycle of length at least 4 or a claw in a graph for which `3free-min-LexBFS` does not generate a proper interval ordering. Remember that a proper interval ordering is an ordering that is an interval ordering as well as a reversed interval ordering.

Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. For a vertex $x$ of $G$, $N_G^+(x)$ denotes the set of neighbours of $x$ that are to the right of $x$ with respect to $\sigma$. Similarly, $N_G^-(x)$ denotes the set of left neighbours of $x$.

**Lemma 11.23** *Let $G = (V, E)$ be a connected graph, and let $\sigma$ be a 3free-min-LexBFS-ordering for $G$. If $\sigma$ is not a proper interval ordering for $G$, then $G$ contains an asteroidal triple, a chordless cycle of length at least 4 or a claw.*

**Proof:** Assume that $\sigma$ is not a proper interval ordering for $G$. Let $z =_{\text{def}} \sigma(n)$, and let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z$. Since $\sigma$ is also a 2free-min-LexBFS-ordering, Lemma 11.17 shows that, if $\sigma$ is not an interval ordering for $G$, $G$ contains an asteroidal triple, a chordless cycle of length at least 4 or a claw. Now, assume that $\sigma$ is an interval ordering for $G$, i.e., $G$ is an interval graph. Since $\sigma$ is not a proper interval ordering, $\sigma$ is not a reversed interval ordering, and there must be a leftmost vertex $u \in V$, its rightmost neighbour $w$, and a leftmost vertex $v$ such that $u \prec_\sigma v \prec_\sigma w$ and $uv \notin E$. If there is a neighbour $x$ of $w$ to the right of $w$ that is not a neighbour of $v$, then $\{u, v, w, x\}$ induces a claw in $G$. Note that, since $\sigma$ is an interval ordering, $vx \notin E$ implies $ux \notin E$. In the remaining part of the proof, we will show that $N_G^+(w) \subseteq N_G(v)$ is not possible, which completes the proof.

We can assume that $k \geq 2$, since otherwise $G$ would be complete (and $\sigma$ a proper interval ordering). Let $a =_{\text{def}} \sigma(1)$. Let $w \in S_i$ for some $i \geq 0$. Since $w$ is the rightmost neighbour of $u$, $u \in S_{i+1}$. We assume $N_G^+(w) \subseteq N_G(v)$. So, $v$ is also in $S_i$, and therefore $i \geq 1$. We first observe that $\text{lm}(w) = u$, since a neighbour of $w$ to the left of $u$ must also be a neighbour of $v$ which implies $u$ being a neighbour of $v$ by $\sigma$ being an interval ordering. Similarly, $\text{lm}(v)$ is strictly to the right of $u$. Since all vertices between $u$ and $w$ are neighbours of $w$, $N_G^-(v) \subseteq N_G(w)$. If there is a shortest $a, z$-path containing $u$ or a vertex from $S_{i+1}$ to the right of $u$, then there is a shortest $a, u$-path of length $k - i - 1$ by the choice of $u$; if there are only shortest $a, z$-paths containing a vertex from $S_{i+1}$ to the left of $u$, then $\text{dist}_G(a, u) = k - i$, since every vertex from $S_{i+1}$ to the left of $u$ is adjacent with $u$. If there is an $a, w$-path of length $k - i$, there must be an $a, w$-path of length $k - i$ containing $u$, since $u = \text{lm}(w)$. It follows that $\text{dist}_G(a, w) = \text{dist}_G(a, u) + 1$. If there is an $a, v$-path of length $k - i$ then there is an $a, u$-path of length $k - i - 1$. It follows that $\text{dist}_G(a, u) < \text{dist}_G(a, w) \leq \text{dist}_G(a, v)$. Let $S_u$ and $S_w$ be the sets of vertices at distance $\text{dist}_G(a, u)$ and $\text{dist}_G(a, w)$ to $a$, respectively. Every vertex in $S_u$ appears to the left of $v$, which follows from arguments above and $\text{lm}(w) = u$, so that $N_G(v) \cap S_u \subseteq N_G(w)$. For every vertex $x \in S_u$, $w \in N_G(x)$ implies $N_G(u) \cap S_w \subseteq N_G(x)$, since $\sigma$ is an interval ordering.

As we have seen, $v \notin S_k$, which means that $\sigma$ on $S_k$ is a proper interval ordering. Since $a$ has a neighbour in $S_{k-1}$, $S_k$ is a clique. If $S_k$ contains two moplexes, $G$ cannot be chordal due to Fact 11.5, and therefore, $a$ is vertex in the only moplex in $S_k$. Furthermore, $S_k$ contains all vertices at distance $\text{diam}(G)$ to $z$. Corollary 11.22 shows that the second sweep of `3free-min-LexBFS` must have started with a vertex from $S_k$, precisely, with a vertex of the moplex contained in $S_k$. Since `min-LexBFS` can number moplex vertices in any order, we conclude that $a$ is the moplex vertex with least priority during the third sweep, hence the vertex that started the second sweep. Now, consider the second sweep of `3free-min-LexBFS`. Before a vertex from $S_u$ is chosen by the algorithm, $v$ and $w$ remain in the leftmost box. $S_u$ contains no vertex that is a neighbour of $v$ but not of $w$. As long as $u$ has not been chosen, $w$ remains in the leftmost box that contains neighbours of $u$. So, when $u$ is finally chosen, the box containing $w$ must be partitioned. Either $v$ and $w$ are still in the same box, then $u$ separates $v$ and $w$, or a previous selection caused a separation of $v$ and $w$. The second sweep therefore assigns high priority to $v$ with respect to $w$. During the third sweep of `3free-min-LexBFS`, $v$ cannot be contained in a box to the left of a box containing $w$ by assumption $N_G^+(w) \subseteq N_G(v)$. If $v$ and $w$ are always contained in the same box, $v$ must be chosen before $w$ due to its higher priority. This contradicts $v \prec_\sigma w$.

■

**Theorem 11.24** *There is a linear-time certifying recognition algorithm for proper interval graphs that outputs a proper interval ordering, an asteroidal triple, a chordless cycle of length at least 4 or a claw.*

**Proof:** A graph is a proper interval graph if and only if every connected component of it is a proper interval graph. Given a graph $G$, apply `3min-LexBFS` to each connected component of $G$. The first paragraph of the proof of Lemma 11.23 describes a linear-time algorithm that verifies whether the generated vertex orderings are proper interval orderings and finds an asteroidal triple, a chordless cycle of length at least 4 or a claw, if one is not a proper interval ordering.

■

With little effort, it can be shown that a $k$min-LexBFS-ordering of a disconnected graph is composed of $k$min-LexBFS-orderings for every of its connected components. So, our recognition algorithm does not have to decompose the input graph into its connected components, but can directly be applied to it.

**Theorem 11.25** *Let $G$ be a connected proper interval graph. A vertex ordering for $G$ is a proper interval ordering for $G$ if and only if it is a 3free-min-LexBFS-ordering for $G$.*

**Proof:** Due to Lemma 11.23 and Theorem 11.12, every 3free-min-LexBFS-ordering for $G$ is a proper interval ordering for $G$. For the converse, let $\sigma$ be a proper interval ordering for $G$, that exists due to Theorem 11.13. Then, $\vec{\sigma}$, the reverse of $\sigma$, is also a proper interval ordering for $G$. Since proper interval orderings for $G$ are min-LexBFS-orderings for $G$, it holds that $\sigma = \texttt{min-LexBFS}^*(G, \vec{\sigma})$ and $\vec{\sigma} = \texttt{min-LexBFS}^*(G, \sigma)$. (We have already seen these arguments in the proof of Theorem 11.20.) Hence, $\sigma = \texttt{3min-LexBFS}^*(G, \vec{\sigma})$, i.e., $\sigma$ is a 3free-min-LexBFS-ordering for $G$.

■

## 11.6 Solving the min-Tri membership problem

The membership problem that we wish to solve in this chapter is defined as follows: given a pair $(G, H)$ of graphs, where $G$ is an AT-free claw-free graph, is $H$ a minimal triangulation of $G$? We will call this problem the *min-Tri membership problem for AT-free claw-free graphs.* A variant of this problem is called the *promise min-Tri membership problem for AT-free claw-free graphs* where we trust that the input graph $G$ is AT-free claw-free. At the end, we will discuss that impact of an alternative representation of $H$, i.e., a representation that is not defined by adjacency lists, on the complexity of the membership problems.

**Theorem 11.26** *There is a linear-time algorithm for solving the promise min-Tri membership problem for AT-free claw-free graphs.*

**Proof:** The central procedure which our algorithm relies on is depicted in Figure 27. This procedure works for connected graphs. Let $(G, H)$ be a pair of graphs where $G$ is connected AT-free claw-free. If $H$ is a minimal triangulation of $G$, then $H$ is a spanning supergraph of $G$, which is verified in lines 2–4 of `MSP_minTri_ATfreeclawfree`. Furthermore, if $H$ is a minimal triangulation of $G$, then $H$ is a chordal AT-free claw-free graph (Theorem 11.14), i.e., a proper interval graph (Theorem 11.12). Hence, `3min-LexBFS` in line 5 generates a proper interval ordering $\sigma$ for $H$ due to Theorem 11.25. If $H$ is a minimal triangulation of $G$, every 2free-min-LexBFS-ordering for $H$ is a 1free-min-LexBFS-ordering for $G$, i.e., a min-LexBFS-ordering for $G$ due to Lemma 11.18. In lines 9–13 of `MSP_minTri_ATfreeclawfree`, it is tested whether $\sigma$ and $\vec{\sigma}$ are min-LexBFS-orderings for $G$. If $\sigma$ passes the test, it is a 2free-min-LexBFS-ordering for $G$, and $(G, \sigma)$ is a minimal triangulation of $G$ due to Theorem 11.20. Since $\sigma$ is an interval ordering for $H$ and if $H$ is a minimal triangulation of $G$, $H = (G, \sigma)$ due to Lemma 11.19. If $H$ is not a minimal triangulation, $(G, \sigma)$ cannot equal $H$. The described procedure is a linear-time algorithm, since every step can be performed in linear time. Note that equality in line 14 can be verified by checking whether $H \subseteq (G, \sigma)$ holds, which can be done in linear time.

Now, let $(G, H)$ be a pair of graphs where $G$ is an arbitrary AT-free claw-free graph. It holds that $H$ is a minimal triangulation of $G$ if and only if there is a 1-to-1 correspondence between the connected components of $G$ and $H$ such that the ones are minimal triangulations of the others. This correspondence can be computed in linear time, if it is possible. Then, `MSP_minTri_ATfreeclawfree` is applied to every

```
MSP_minTri_ATfreeclawfree($G$, $H$) returns Boolean:
begin
   if (($V(G) \neq V(H)$) or ($E(G) \not\subseteq E(H)$)) then
      return false
   end if;
   let $\sigma =_{\mathrm{def}}$ 3min-LexBFS($H$);
   if ($\sigma$ is not proper interval ordering for $G$) then
      return false
   end if;
   let $\sigma' =_{\mathrm{def}}$ min-LexBFS$^*$($G$, $\vec{\sigma}$);
   let $\sigma'' =_{\mathrm{def}}$ min-LexBFS$^*$($G$, $\sigma$);
   if (($\sigma \neq \sigma'$) or ($\vec{\sigma} \neq \sigma''$) then
      return false
   end if;
   return ($H = (G, \sigma)$)
end.
```

**Figure 27** The linear-time algorithm that solves the promise min-Tri membership problem for connected AT-free claw-free graphs.

pair of connected components, and we obtain overall linear time.

■

For solving the min-Tri membership problem for AT-free claw-free graphs, we have to know whether input graph $G$ is AT-free claw-free. This problem was solved by Hempel and Kratsch.

**Theorem 11.27 (Hempel and Kratsch, [40])**
*There is an algorithm that recognises AT-free claw-free graphs in time $\mathcal{O}(n^{2.376})$.*

The time of the recognition algorithm is determined by multiplication of two binary square matrices and recognition of triangle-free graphs. The authors additionally showed that any AT-free claw-free graphs recognition algorithm with worst case time $o(n^{2.376})$ would improve the time for recognition of triangle-free graphs [40], which is a challenging task, since finding triangles is a well-studied problem.

**Theorem 11.28** *The min-Tri membership problem for AT-free claw-free graphs can be solved in time $\mathcal{O}(n^{2.376})$.*

**Proof:** Let $(G, H)$ be a pair of graphs. The pair has to be accepted if and only if $G$ is an AT-free claw-free graph and $H$ is a minimal triangulation of $G$. The former condition can be tested in time $\mathcal{O}(n^{2.376})$ due to Theorem 11.27, the latter condition can be tested in linear time due to Theorem 11.26.

■

We want to discuss a variant of algorithm `MSP_minTri_ATfreeclawfree`, that is called `MSP_minTri_ATfreeclawfree_short` and depicted in Figure 28. It varies from `MSP_minTri_ATfreeclawfree` only in the condition at the end of the procedure: it is not verified whether $\vec{\sigma}$ is a 2free-min-LexBFS-ordering. We will show that this variant is nevertheless correct.

**Theorem 11.29** *Let $G = (V, E)$ be a connected AT-free claw-free graph, and let $H$ be graph. Then,* `MSP_minTri_ATfreeclawfree_short` *accepts input $(G, H)$ if and only if $H$ is a minimal triangulation of $G$.*

**Proof:** We reconsider the proof of Theorem 11.26 and observe that it remains to show that $(G, \sigma)$ is a minimal triangulation of $G$, if $H = (G, \sigma)$ holds. Observe that, in line 12 of the algorithm, we know that $\sigma$ is a min-LexBFS-ordering for $G$, but we do not know whether $\sigma$ is a 2free-min-LexBFS-ordering for $G$. A thorough analysis of the proof of Lemma 11.15 shows that $(G, \sigma)$ is a minimal triangulation of $G$, if $\sigma$ is a min-LexBFS-ordering for $G$ that ends with a moplex vertex. So, it suffices to show for our result that $z = \sigma(n)$ is a moplex vertex of $G$. Let $H' =_{\text{def}} (G, \sigma)$, and let $H = H'$. Let $S_0, \ldots, S_k$ be the BFS-levels of $G$ with root vertex $z$. The proof of Lemma 11.15 shows that the BFS-levels of $G$ and $H'$ with root $z$ are equal. Hence, $N_G(z) = N_{H'}(z) = N_H(z)$. Suppose $k \leq 1$. Then, $z$ is a universal vertex in $G$ and $H$,

```
MSP_minTri_ATfreeclawfree_short($G$, $H$) returns Boolean:
begin
   if (($V(G) \neq V(H)$) or ($E(G) \not\subseteq E(H)$)) then
      return false
   end if;
   let $\sigma =_{\text{def}}$ 3min-LexBFS($H$);
   if ($\sigma$ is not proper interval ordering for $G$) then
      return false
   end if;
   let $\sigma' =_{\text{def}}$ min-LexBFS*($G$, $\vec{\sigma}$);
   if ($\sigma \neq \sigma'$) then
      return false
   end if;
   return ($H = (G, \sigma)$)
end.
```

**Figure 28** Another linear-time algorithm that solves the promise min-Tri membership problem for connected AT-free claw-free graphs.

and since $\sigma$ is a proper interval ordering for $H$, $H$ is complete. If $G$ is not complete there is a vertex in $G$ that is not adjacent with $\sigma(1)$ due to the moplex property of $\sigma(1)$. Hence, $H'$ is not complete, $H \neq H'$, and therefore $k \geq 2$. Let $M \subset S_1$ be the set of vertices from $S_1$ without a neighbour in $S_2$. By the definition of $(G, \sigma)$ (and since we assume $H = H'$), the vertices in $M$ do not have neighbours in $S_2$ in $H$. Since $\sigma$ is a proper interval ordering for $H$, all vertices in $M$ are to the right of all vertices in $S_1 \setminus M$ in $\sigma$. Let $u$ be the vertex from $S_1$ leftmost with respect to $\sigma$. If $u$ is neighbour of every vertex from $M$ in $G$ and by the farther right neighbour property of $\sigma$, $M$ is a clique in $G$. Hence, $M \cup \{z\}$ is a module, even a maximal clique-module of $G$. Finally, $S_2$ is a clique in $G$, so that $S_1 \setminus M$ is a minimal separator of $G$, which means that $M$ is a moplex of $G$, and $z$ is a moplex vertex of $G$.

■

At the end of this section, we want to discuss the impact of representation on the time complexity of algorithm `MSP_minTri_ATfreeclawfree`. In the original form of the algorithm, both graphs $G$ and $H$ are given by adjacency lists. We know that $H$ can have significantly more edges than $G$. So, the representation of $H$ can become more succinct, if we require $H$ to be given by an interval model without interval that is properly contained in another. We assume that the interval endpoints are from the set $\{1, \ldots, 2n\}$, which is a reasonable assumption. In time $\mathcal{O}(n)$, a proper interval ordering $\sigma$ for $H$ can be computed, and in linear time with respect to $G$, it can be verified whether $\sigma$ and $\vec{\sigma}$ are min-LexBFS-orderings for $G$. Given the interval model, adjacency can be tested in constant time, so that the subgraph relationship $G \subseteq H$ can be tested in linear time with respect to $G$. It remains to verify $H = (G, \sigma)$. It is clear that this equality cannot be tested by verification of each edge. However, $\sigma$ is an interval ordering for $H$ and for $(G, \sigma)$ (proof of Lemma 11.15), so that it suffices to compare leftmost neighbours. This is done in linear time with respect to $G$.

## 11.7 Algorithmic applications

In contrast to the graph classes considered before (the classes of $2K_2$-free graphs and permutation graphs), all three problems, Treewidth, Minimum Fill-in and Bandwidth, are NP-complete for AT-free claw-free graphs (Theorems 8.23, 8.24 and 8.25), since co-bipartite graphs are AT-free claw-free. We will present a simple and fast algorithm for approximating the bandwidth of AT-free claw-free graphs.

A number of algorithms for approximating the bandwidth of AT-free graphs have been proposed in the literature. They are based on the following interesting bounds. By $G^2$, we denote the *square* of graph $G = (V, E)$, i.e., the graph on vertex set $V$ that has an edge between vertices $u$ and $v$ if and only if $u$ and $v$ are connected by a path of length 1 or 2 in $G$. The square of $G$ can be computed by squaring the matrix that emerges from the adjacency matrix of $G$ by adding 1's on the main diagonal. The computed square of the matrix represents the adjacency matrix of $G$ where entries on the main diagonal are ignored and edges are represented by non-zero entries. The square of a matrix can be computed in time $\mathcal{O}(n^{2.376})$ [18]. By $\Delta(G)$, the largest number of neighbours of a vertex of $G$ is denoted.

**Lemma 11.30 (Fomin and Golovach, [28])**
*Let $G$ be an AT-free graph. Then, $\frac{1}{4}\Delta(G^2) \leq \mathrm{bw}(G) \leq \Delta(G^2)$.*

**Corollary 11.31** *There is a 4-approximation algorithm for the bandwidth problem on AT-free graphs that runs in time $\mathcal{O}(n^{2.376})$.*

The time bound of the algorithm of Corollary 11.31 is mainly determined by the computation of the square of the input graph. So, the authors asked whether the maximal degree of the square of a graph can be computed faster. For our purposes, it suffices to find an algorithm for squares of AT-free claw-free graphs. Another approach involves minimal triangulations.

**Lemma 11.32 (Kloks, Kratsch and Müller, [50])**
*Let $G$ be an AT-free graph, and let $H$ be a minimal triangulation of $G$. Then,* $\mathrm{bw}(G) \leq \mathrm{bw}(H) \leq 2 \cdot \mathrm{bw}(G)$.

**Corollary 11.33** *There is a 2-approximation algorithm for the bandwidth problem on AT-free graphs that runs in time $\mathcal{O}(nm)$.*

A third approach was used by the authors of [50] to obtain a fast 4-approximation algorithm; it is based on finding a special spanning tree that approximates shortest paths between every pair of vertices.

**Theorem 11.34 (Kloks, Kratsch and Müller, [50])**
*There is a 4-approximation algorithm for the bandwidth problem on AT-free graphs that runs in time $\mathcal{O}(m + n \log n)$.*

Our algorithm for AT-free claw-free graphs is inspired by Lemma 11.32. So, we have to compute the bandwidth of a minimal triangulation of an AT-free claw-free graph. Due to Theorem 11.14, these graphs are proper interval graphs, and the following result shows that the bandwidth of a proper interval graph is easy to determine. For a graph $G = (V, E)$, the *proper pathwidth* of $G$, denoted as $\mathrm{ppw}(G)$, is one less than the smallest clique number among all triangulations of $G$ that are proper interval graphs.

**Lemma 11.35 (Kaplan, Shamir and Tarjan, [46])**
*Let $G = (V, E)$ be a graph. Then,* $\mathrm{bw}(G) = \mathrm{ppw}(G)$.

**Corollary 11.36** *Let $G$ be a proper interval graph. Then,* $\mathrm{bw}(G) = \omega(G) - 1$.

**Proof:** Since $G$ is a proper interval graph, $G$ is the only minimal triangulation of $G$. Hence, the treewith of $G$ equals the proper pathwidth of $G$, and the treewidth of a chordal graph is one less than its clique number. So, $\mathrm{ppw}(G) = \omega(G) - 1$, and the claim follows by Lemma 11.35.

■

**Theorem 11.37** *There is a linear-time 2-approximation algorithm for the bandwidth problem on AT-free claw-free graphs.*

**Proof:** Let $G = (V, E)$ be an AT-free claw-free graph. In linear time, `2min-LexBFS` computes a minimal interval elimination ordering $\sigma$ for $G$ due to Lemma 11.15. Then, $(G, \sigma)$ is a minimal triangulation of $G$ and a proper interval graph due to Theorem 11.14. In linear time, an interval model for $(G, \sigma)$ can be computed, and in time $\mathcal{O}(n)$, the clique number $c$ of $(G, \sigma)$ can be determined. Due to Corollary 11.36, the bandwidth of $(G, \sigma)$ equals $c - 1$, and due to Lemma 11.32, $\mathrm{bw}(G) \leq c - 1 \leq 2 \cdot \mathrm{bw}(G)$.

■

**Theorem 11.38 (Parra and Scheffler, [69])**
*Let $G$ be an AT-free claw-free graph. Then,* $\mathrm{ppw}(G) = \mathrm{tw}(G)$.

**Proof:** For every graph $G$, $\mathrm{ppw}(G) \geq \mathrm{tw}(G)$. For an AT-free claw-free graph $G$, every minimal triangulation is a proper interval graph, hence $\mathrm{tw}(G) \geq \mathrm{ppw}(G)$.

■

**Corollary 11.39** *There is a linear-time 2-approximation algorithm for the treewidth problem on AT-free claw-free graphs.*

## 11.8 Interesting problems

In the previous sections, we proved a number of results about min-LexBFS-orderings for arbitrary and special graphs. It seems that this field of research is rich. For example, we saw that 2free-min-LexBFS-orderings for proper interval graphs are interval orderings (Lemma 11.17), hence perfect elimination schemes. Is there a number $k$ such that $k$-min-LexBFS-orderings of chordal graphs are perfect elimination schemes?

We also saw that, for $G$ a connected AT-free claw-free graph, $H$ a minimal triangulation of $G$ and $\sigma$ a proper interval ordering for $H$, $\sigma$ and $\vec{\sigma}$ are min-LexBFS-orderings for $G$ (Lemma 11.18). It holds that $\sigma = $ `2min-LexBFS`$^*(G, \sigma)$. This inspires the definition of fixed points of `min-LexBFS`: Let $G = (V, E)$ be a graph, and let $\sigma$ be a vertex ordering for $G$. We say that $\sigma$ is a *fixed point* of `2min-LexBFS`$^*$ for $G$, if $\sigma = $ `2min-LexBFS`$^*(G, \sigma)$ holds. What are these fixed points for arbitrary or special graphs? Do they provide structural information about the graph? Do they always exist, and can a fixed point be generated fast? Is there a number $k_G$ for graph $G$ such that $k_G$`min-LexBFS`$(G)$ is a fixed point?

We have mentioned that `min-LexBFS` and `LexBFS` are related algorithms. Are LexBFS- and min-LexBFS-orderings related? Are there (non-trivial) graph classes for which LexBFS-orderings are min-LexBFS-orderings or vice versa? Analogous to the definition of $k$`min-LexBFS`, algorithms involving `LexBFS` and `min-LexBFS` can be defined. What can be said about them?

For $2K_2$-free graphs and permutation graphs, we characterised minimal triangulations in a way that enabled "efficient" enumeration of all minimal triangulations; efficiency shall mean that the algorithm runs fast per minimal triangulation. In case of AT-free claw-free graphs, such a fast algorithm is not immediate. Is there a way to efficiently guide `2free-min-LexBFS` to generate all 2free-min-LexBFS-orderings for an AT-free claw-free graph?

## Bibliography

[1] E. Allender, M. Ogihara, *Relationships among PL, #L, and the Determinant*, Informatique théorique et Applications 30, No. 1, pp. 1–21, 1996.

[2] E. Allender, K. Reinhardt, Sh. Zhou, *Isolation, Matching, and Counting Uniform and Nonuniform Upper Bounds*, Journal of Computer and System Sciences 59, pp. 164–181, 1999.

[3] C. Àlvarez, J.L. Balcázar, B. Jenner, *Adaptive Logspace Reducibility and Parallel Time*, Mathematical Systems Theory 28, pp. 117-140, 1995.

[4] St. Arnborg, D.G. Corneil, A. Proskurowski, *Complexity of finding embeddings in a $k$-tree*, SIAM Journal on Algebraic and Discrete Methods 8, pp. 277–284, 1987.

[5] A. Berry, J.-P. Bordat, *Separability generalizes Dirac's theorem*, Discrete Applied Mathematics 84, pp. 43–53, 1998.

[6] J.R.S. Blair, P. Heggernes, J.A. Telle, *A Practical Algorithm for Making Filled Graphs Minimal*, Theoretical Computer Science 250, pp. 125–141, 2001.

[7] M. Blum, *A Machine-Independent Theory of the Complexity of Recursive Functions*, Journal of the Association for Computing Machinary 14, pp. 322–336, 1967.

[8] H.L. Bodlaender, T. Kloks, D. Kratsch, *Treewidth and Pathwidth of Permutation Graphs*, SIAM Journal on Discrete Mathematics 8, pp. 606–616, 1995.

[9] H.L. Bodlaender, T. Kloks, D. Kratsch, H. Müller, *Treewidth and minimum fill-in on d-trapezoid graphs*, Journal of Graph Algorithms and Applications 2, No. 3, pp. 1–23, 1998.

[10] H.L. Bodlaender, R.H. Möhring, *The pathwidth and treewidth of cographs*, SIAM Journal on Discrete Mathematics 6, pp. 181–188, 1993.

[11] K.S. Booth, G.S. Lueker, *Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms*, Journal of Computer and System Sciences 13, pp. 335–379, 1976.

[12] A. Borodin, S. Cook, N. Pippenger, *Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines*, Information and Control 58, pp. 113–136, 1983.

[13] V. Bouchitté, I. Todinca, *Treewidth and Minimum Fill-in: Grouping the Minimal Separators*, SIAM Journal on Computing 31, pp. 212–232, 2001.

[14] A. Brandstädt, V.B. Le, J.P. Spinrad, *Graph classes: a survey*, SIAM monographs on discrete mathematics and applications, 1999.

[15] A. Bretscher, D.G. Corneil, M. Habib, Chr. Paul, *A Simple Linear Time LexBFS Cograph Recognition Algorithm*, Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2003, Lecture Notes in Computer Science 2880, pp. 119–130, 2003.

[16] P. Buneman, *A Characterisation of Rigid Circuit Graphs*, Discrete Mathematics 9, pp. 205–212, 1974.

[17] St.A. Cook, *The complexity of theorem-proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1971.

[18] D. Coppersmith, Sh. Winograd, *Matrix Multiplication via Arithmetic Progressions*, Journal of Symbolic Computation 9, pp. 251–280, 1990.

[19] D.G. Corneil, *A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs*, Discrete Applied Mathematics 138, pp. 371–379, 2004.

[20] D.G. Corneil, H. Kim, S. Natarajan, St. Olariu, A.P. Sprague, *Simple linear time recognition of unit interval graphs*, Information Processing Letters 55, pp. 99–104, 1995.

[21] D.G. Corneil, St. Olariu, L. Stewart, *Asteroidal triple-free graphs*, SIAM Journal on Discrete Mathematics 10, pp. 399–430, 1997.

[22] D.G. Corneil, St. Olariu, L. Stewart, *The ultimate interval graph recognition algorithm? (Extended abstract)*, Proceedings of the Nineth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'98, pp. 175–180, 1998.

[23] E. Dahlhaus, *Minimal Elimination Ordering Inside a Given Chordal Graph*, Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG'97, Lecture Notes in Computer Science 1335, pp. 132–143, 1997.

[24] X. Deng, P. Hell, J. Huang, *Linear-time representation algorithm for proper circular-arc graphs and proper interval graphs*, SIAM Journal on Computing 25, pp. 390–403, 1996.

[25] G.A. Dirac, *On rigid circuit graphs.*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 25, pp. 71–76, 1962.

[26] M. Farber, *On diameters and radii of bridged graphs*, Discrete Mathematics 73, pp. 249–260, 1989.

[27] St. Földes, P.L. Hammer, *Split graphs*, 8th South-Eastern Conference on Combinatorics, Graph Theory and Computing, Louisiana State University, Baton Rouge, Louisiana, Congressus Numerantium 19, pp. 311–315, 1977.

[28] F.V. Fomin, P.A. Golovach, *Interval degree and bandwidth of a graph*, Discrete Applied Mathematics 129, pp. 345–359, 2003.

[29] M.L. Fredman, J. Komlós, E. Szemerédi, *Storing a Sparse Table with $\mathcal{O}(1)$ Worst Case Access Time*, Journal of the Association for Computing Machinery 31, pp. 438–544, 1984.

[30] T. Gallai, *Transitiv orientierbare Graphen*, Acta Mathematica Academiae Scientiarum Hungaricae 18, pp. 25–66, 1967.

[31] M.R. Garey, D.S. Johnson, *Computers and Intractability. A Guide to the Theory of $NP$-Completeness*, W.H. Freeman and Company, 1979.

[32] F. Gavril, *The Intersection Graphs of Subtrees in Trees Are Exactly the Chordal Graphs*, Journal of Combinatorial Theory (B) 16, pp. 47–56, 1974.

[33] P.C. Gilmore, A.J. Hoffman, *A Characterization of Comparability Graphs and of Interval Graphs*, Canadian Journal of Mathematics 16, pp. 539–548, 1964.

[34] Chr. Glasser, *private communication*, 2003.

[35] M.Ch. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[36] M.Ch. Golumbic, Cl. Monma, W.T. Trotter, Jr., *Tolerance Graphs*, Discrete Applied Mathematics 9, pp. 157–170, 1984.

[37] M. Habib, R. McConnell, Chr. Paul, L. Viennot, *Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing*, Theoretical Computer Science 234, pp. 59–84, 2000.

[38] M. Habib, R.H. Möhring, *Treewidth of cocomparability graphs and a new order-theoretic parameter*, Technical report 336/1992, Fachbereich 3 Mathematik, Technische Universität Berlin, 1992.

[39] P. Hell, J. Huang, *Certifying LexBFS Recognition Algorithms for Proper Interval Graphs and Proper Interval Bigraphs*, SIAM Journal on Discrete Mathematics 18, pp. 554–570, 2005.

[40] H. Hempel, D. Kratsch, *On claw-free asteroidal triple-free graphs*, Discrete Applied Mathematics 121, pp. 155–180, 2002.

[41] C.M. Herrera de Figueiredo, J. Meidanis, C. Picinin de Mello, *A linear-time algorithm for proper interval graph recognition*, Information Processing Letters 56, pp. 179–184, 1995.

[42] W. Hesse, E. Allender, D.A. Mix Barrington, *Uniform constant-depth threshold circuits for division and iterated multiplication*, Journal of Computer and System Sciences 65, pp. 695–716, 2002.

[43] D. Hilbert, *Mathematische Probleme*, Nachrichten von der Königlichen Gesellschaft der Wissenschaften zu Göttingen 1900, Göttingen, pp. 253–297, 1900.

[44] L. Ibarra, *Fully dynamic algorithms for chordal graphs and split graphs*, Technical report, University of Victoria, 2000.

[45] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM Journal on Computing 17, pp. 935–938, 1988.

[46] H. Kaplan, R. Shamir, R.E. Tarjan, *Tractability of parameterized completion problems on chordal and interval graphs: Minimum Fill-in and Physical Mapping*, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS 1994, pp. 780–791, IEEE Computer Society Press, 1994.

[47] T. Kloks, *Treewidth—Computations and Approximations*, Lecture Notes in Computer Science 842, Springer, 1994.

[48] T. Kloks, D. Kratsch, *Listing all minimal separators of a graph*, SIAM Journal on Computing 27, pp. 605–613, 1998.

[49] T. Kloks, D. Kratsch, H. Müller, *Approximating the Bandwidth for Asteroidal Triple-Free Graphs*, Proceedings of the 3rd Annual European Symposium on Algorithms, ESA'95, Lecture Notes in Computer Science 979, pp. 434–447, 1995.

[50] T. Kloks, D. Kratsch, H. Müller, *Approximating the bandwidth for AT-free graphs*, Journal of Algorithms 32, pp. 41–57, 1999.

[51] T. Kloks, D. Kratsch, H. Müller, *Finding and counting small induced subgraphs efficiently*, Information Processing Letters 74, pp. 115–121, 2000.

[52] T. Kloks, D. Kratsch, J. Spinrad, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theoretical Computer Science 175, pp. 309–335, 1997.

[53] D. Kratsch, R.M. McConnell, K. Mehlhorn, J.P. Spinrad, *Certifying Algorithms for Recognizing Interval Graphs and Permutation Graphs*, Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003, pp. 709–716, 2003.

[54] D. Kratsch, L. Stewart, *Domination on cocomparability graphs*, SIAM Journal on Discrete Mathematics 6, pp. 400–417, 1993.

[55] J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pp. 527–631, Elsevier, 1990.

[56] C.G. Lekkerkerker, J.Ch. Boland, *Representation of finite graphs by a set of intervals on the real line*, Fundamenta Mathematicae 51, pp. 45–64, 1962.

[57] M. Liśkiewicz, M. Ogihara, S. Toda, *The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes*, Theoretical Computer Science 304, pp. 129–156, 2003.

[58] P.J. Looges, St. Olariu, *Optimal greedy algorithms for indifference graphs*, Computers & Mathematics with Applications 25, pp. 15–25, 1993.

[59] Y.V. Matiyasevich, *Hilbert's Tenth Problem*, The MIT Press, 1993.

[60] R.M. McConnell, J.P. Spinrad, *Modular decomposition and transitive orientation*, Discrete Mathematics 201, pp. 189–241, 1999.

[61] P. McKenzie, K.W. Wagner, *The Complexity of Membership Problems for Circuits over Sets of Natural Numbers*, Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2003, Lecture Notes in Computer Science 2607, Springer, pp. 571–582, 2003.

[62] K. Mehlhorn, St. Näher, *The LEDA Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 1999.

[63] D. Meister, *Minimale Triangulationen AT-freier Graphen*, Diploma thesis, Friedrich-Schiller-Universität Jena, Germany, 2000.

[64] D. Meister, *The structure of separator graphs and efficient algorithms for minimal triangulation problems of permutation graphs*, Technical report 361, Institut für Informatik, Bayerische Julius-Maximilians-Universität Würzburg, 2005.

[65] R.H. Möhring, *Triangulating graphs without asteroidal triples*, Discrete Applied Mathematics 64, pp. 281–287, 1996.

[66] St. Olariu, *An optimal greedy heuristic to color interval graphs*, Information Processing Letters 37, pp. 21–25, 1991.

[67] B.S. Panda, S.K. Das, *A linear time recognition algorithm for proper interval graphs*, Information Processing Letters 87, pp. 153–161, 2003.

[68] Ch.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[69] A. Parra, P. Scheffler, *Characterizations and algorithmic applications of chordal graph embeddings*, Discrete Applied Mathematics 79, pp. 171–188, 1997.

[70] A. Pnueli, A. Lempel, Sh. Even, *Transitive orientation of graphs and identification of permutation graphs*, Canadian Journal of Mathematics 23, pp. 160–175, 1971.

[71] E. Prisner, *Graphs with Few Cliques*, Graph Theory, Combinatorics, and Applications: Proceedings of the Seventh Quadrennial International Conference on the Theory and Applications of Graphs, pp. 945–956, John Wiley and Sons, Inc., 1995.

[72] O. Reingold, *Undirected ST-Connectivity in Log-Space*, Electronic Colloquium on Computational Complexity, ECCC, report no. 94, 2004.

[73] F.S. Roberts, *Indifference graphs*, in: F. Harary (Ed.), Proof techniques in graph theory, pp. 139–146, Academic Press, New York, 1969.

[74] N. Robertson, P.D. Seymour, *Graph Minors. II. Algorithmic Aspects of Tree-Width*, Journal of Algorithms 7, pp. 309–322, 1986.

[75] D.J. Rose, *Triangulated Graphs and the Elimination Process*, Journal of Mathematical Analysis and Applications 32, pp. 597–609, 1970.

[76] D.J. Rose, R.E. Tarjan, G.S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Jounal on Computing 5, pp. 266–283, 1976.

[77] W.L. Ruzzo, J. Simon, M. Tompa, *Space-Bounded Hierarchies and Probabilistic Computations*, Journal of Computer and System Sciences 28, pp. 216–230, 1984.

[78] W.J. Savitch, *Relationships Between Nondeterministic and Deterministic Tape Complexities*, Journal of Computer and System Sciences 4, pp. 177–192, 1970.

[79] L.J. Stockmeyer, A.R. Meyer, *Word Problems Requiring Exponential Time*, Proceedings of the ACM Symposium on the Theory of Computation, pp. 1–9, 1973.

[80] R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Informatica 26, pp. 279–284, 1988.

[81] R.E. Tarjan, M. Yannakakis, *Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs*, SIAM Journal on Computing 13, pp. 566–579, 1984.

[82] L.G. Valiant, *The Complexity of Enumeration and Reliability Problems*, SIAM Journal on Computing 8, pp. 410–421, 1979.

[83] H. Vollmer, *Introduction to Circuit Complexity*, Springer, 1999.

[84] K. Wagner, *The Complexity of Problems Concerning Graphs with Regularities*, Proceedings of the 11th International Symposium on Mathematical Fondations of Computer Science, MFCS 1984, Lecture Notes in Computer Science 176, Springer, pp. 544–552, 1984.

[85] J.R. Walter, *Representations of Chordal Graphs as Subtrees of a Tree*, Journal of Graph Theory 2, pp. 265–267, 1978.

[86] H. Wasserman, M. Blum, *Software Reliability via Run-Time Result-Checking*, Journal of the ACM 44, pp. 826–849, 1997.

[87] K. Yang, *Integer Circuit Evaluation Is PSPACE-Complete*, Journal of Computer and System Sciences 63, pp. 288–303, 2001.

[88] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM Journal on Algebraic and Discrete Methods 2, pp. 77–79, 1981.

## Mathematical symbolism (a list)

### Number sets

| | |
|---|---|
| $\mathbb{N} =_{\text{def}} \{0, 1, 2, \ldots\}$ | set of natural numbers |
| $\mathbb{Z} =_{\text{def}} \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ | set of integer numbers |
| $\mathbb{R}$ | set of real numbers |

### Sets

| | |
|---|---|
| $\subset$ | proper inclusion relation |
| $\subseteq$ | inclusion relation |
| $\cup$ | union operation |
| $\cap$ | intersection operation |
| $\overline{\phantom{x}}$ | complementation operation |
| $\setminus$ | set subtraction operation |
| $\triangle$ | symmetric difference |

Let $A$ be a set.

| | |
|---|---|
| $A^2 =_{\text{def}} A \times A =_{\text{def}} \{(a, a') : a, a' \in A\}$ | Cartesian set product |

### Functions

| | |
|---|---|
| $a = \log c \iff c = 2^a$ | logarithm function |
| $\text{id}(n) =_{\text{def}} n$ | identity function |
| $\lfloor r \rfloor$ | largest integer not greater than $r$ |

Let $A \subseteq \mathbb{N}$ be a set.

| | |
|---|---|
| $m = \max A \iff m \in A$ and $m \geq n$ for every $n \in A$ | maximum of $A$ |

Functions can be *total*, *injective*, *surjective* and *bijective.*

### Relations

Let $a, b, c \in \mathbb{N}$.

$a \equiv c \pmod{b} \iff \exists k \in \mathbb{N} : |c - a| \cdot k = b$

### Logics

| | |
|---|---|
| $\equiv$ | logical semantic equivalence |
| $\wedge$ | logical 'and' |
| $\vee$ | logical 'or' |
| $\neg$ | logical 'not' |
| $\exists$ | existential quantifier |
| $\forall$ | universal quantifier |

## Graphs

Let $G$, $H$ be graphs, let $G'$ be a directed graph, let $u$ and $v$ be vertices of $G$, let $e$ be an edge of $G$, let $A$ be a set of vertices of $G$, let $F$ be a set of edges of $G$.

| | |
|---|---|
| $V(G)$ | vertex set of $G$ |
| $E(G)$ | edge set of $G$ |
| $V(G')$ | vertex set of $G'$ |
| $A(G')$ | arc set of $G'$ |
| $\subset$ | proper supergraph/subgraph relation |
| $\subseteq$ | supergraph/subgraph relation |
| $\cong$ | isomorphism relation |
| $G[A]$ | subgraph of $G$ induced by $A$ |
| $G-e$ | deleting edge $e$ |
| $G+e$ | adding edge $e$ |
| $G-u$ | deleting vertex $u$ |
| $G+u$ | adding vertex $u$ |
| $G \cup F$ | adding set $F$ of edges |
| $G \cup H$ | disjoint union of $G$ and $H$ |
| $N_G(u)$ | (open) neighbourhood of $u$ |
| $N_G[u]$ | closed neighbourhood of $u$ |
| $N_G(A)$ | (open) neighbourhood of $A$ |
| $N_G[A]$ | closed neighbourhood of $A$ |
| $\mathrm{dist}_G(u,v)$ | distance of $u$ and $v$ |
| $\mathrm{diam}(G)$ | diameter of $G$ |
| $\mathrm{bw}(G)$ | bandwidth of $G$ |
| $\mathrm{mfi}(G)$ | minimum fill-in of $G$ |
| $\mathrm{ppw}(G)$ | proper pathwidth of $G$ |
| $\mathrm{tw}(G)$ | treewidth of $G$ |
| $\alpha(G)$ | independent set number |
| $\omega(G)$ | clique number |

## Complexity theory

| | |
|---|---|
| $\leq_m^{\mathrm{L}}$ | logarithmic-space many-one reducibility |
| $\equiv_m^{\mathrm{L}}$ | logarithmic-space many-one isomorphism |
| $\leq_m^{\mathrm{NL}}$ | nondeterministic logarithmic-space many-one reducibility |

**List of (decision) problems**

Remark: The cited page numbers refer to the first occurrence of the problems, where they are defined

| | | |
|---|---|---|
| ACYC | acyclicness problem | p. 33 |
| ACYCGAP | graph accessibility problem for acyclic graphs | p. 34 |
| ACYCGIP | graph inaccessibility problem for acyclic graphs | p. 34 |
| $\mathrm{aM}_{tm}(\mathcal{O})$ | bounded exact membership problem for recurrent $\mathcal{O}$-systems | p. 68 |
| BANDWIDTH | bandwidth problem | p. 149 |
| DEPTH | depth of a vertex | p. 45 |
| DIOPHANTINE | satisfiability of Diophantine equations | p. 128 |
| EMPTY$(\mathcal{O})$ | emptyness problem for recurrent $\mathcal{O}$-systems | p. 89 |
| $\circ$-EMPTY$(\mathcal{O})$ | special emptyness problem for recurrent $\mathcal{O}$-systems | p. 89 |
| GAP | graph accessibility problem | p. 33 |
| GIP | graph inaccessibility problem | p. 33 |
| $\mathrm{M}_{ex}(\mathcal{O})$ | existential membership problem for recurrent $\mathcal{O}$-systems | p. 67 |
| $\mathrm{M}_{ex}(\mathrm{gr}, \oplus, \otimes, 1)$ | existential membership problem for recurrent $\{\mathrm{gr}, \oplus, \otimes, 1\}$-systems | p. 128 |
| $\mathrm{M}_{tm}(\mathcal{O})$ | exact membership problem for recurrent $\mathcal{O}$-systems | p. 67 |
| $\mathrm{MC}(\mathcal{O})$ | membership problem for arithmetical $\mathcal{O}$-circuits | p. 73 |
| MINIMUM FILL-IN | minimum fill-in problem | p. 149 |
| NOW$(\alpha)$ | number of walks in a graph | p. 44 |
| NOMW$(\alpha)$ | number of walks in a marked graph | p. 44 |
| POM$(\alpha)$ | element of a power of a matrix | p. 48 |
| QBF | true quantified Boolean formula problem | p. 95 |
| RESIDUEOFWALK | residue class of a walk | p. 53 |
| SAT | satisfiability of Boolean expressions | p. 55 |
| 3-SAT | satisfiability of Boolean 3-expressions | p. 55 |
| SET-SCE | satisfiability of a set-system of congruence equations | p. 56 |
| SORSR | simultaneous-ones ring-shift registers problem | p. 58 |
| TREEWIDTH | treewidth problem | p. 148 |
| WALK | existence of a walk | p. 45 |
| xNOW$(\alpha)$ | exact number of walks in a graph | p. 44 |
| xNOMW$(\alpha)$ | exact number of walks in a marked graph | p. 44 |
| xPOM$(\alpha)$ | exact element of a power of a matrix | p. 48 |

# Index