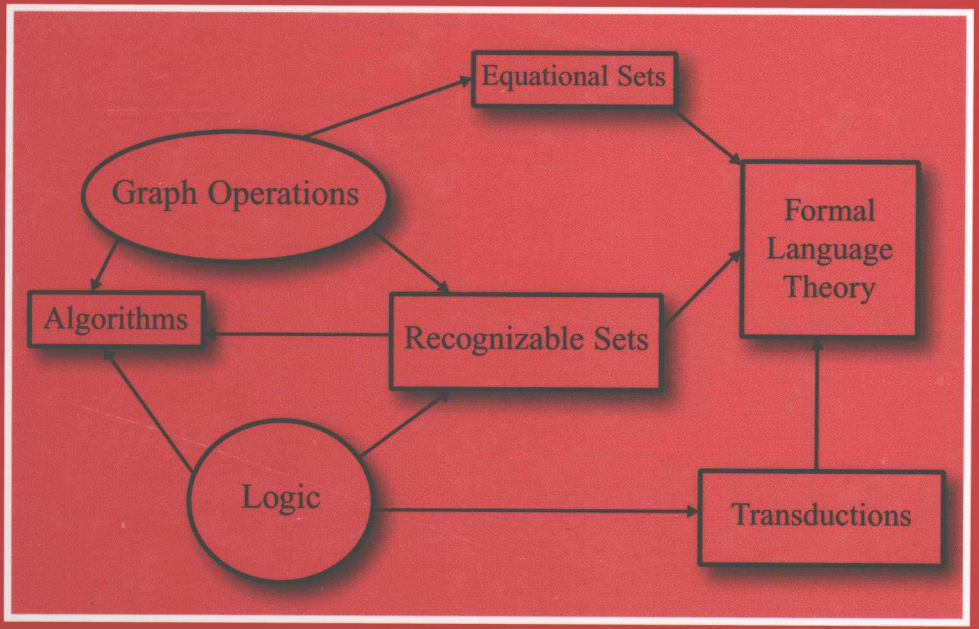
Encyclopedia of Mathematics and Its Applications 138

# GRAPH STRUCTURE AND MONADIC SECOND-ORDER LOGIC

## A Language-Theoretic Approach

Bruno Courcelle and Joost Engelfriet



ENCYCLOPEDIA OF MATHEMATICS AND ITS APPLICATIONS



# *Graph Structure and Monadic Second-Order Logic*

## A Language-Theoretic Approach

BRUNO COURCELLE
*Université de Bordeaux*

JOOST ENGELFRIET
*Universiteit Leiden*



CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Mexico City

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521898331



First published 2012

Printed in the United Kingdom at the University Press, Cambridge

*A catalogue record for this publication is available from the British Library*

*Library of Congress Cataloguing in Publication data*
Courcelle, B.
Graph structure and monadic second-order logic : a language-theoretic approach / Bruno Courcelle, Joost Engelfriet.
p. cm. – (Encyclopedia of mathematics and its applications ; 138)
Includes bibliographical references and index.
ISBN 978-0-521-89833-1 (hardback)
1. Logic, Symbolic and mathematical – Graphic methods. I. Engelfriet, Joost. II. Title.
QA9.C748 2012
511.3–dc23 2012008159

ISBN 978-0-521-89833-1 Hardback

Additional resources for this publication at www.labri.fr/perso/courcell/TheBook.html

# GRAPH STRUCTURE AND MONADIC SECOND-ORDER LOGIC

A Language-Theoretic Approach

The study of graph structure has advanced significantly in recent years: finite graphs can now be described algebraically, enabling them to be constructed out of more basic elements. One can obtain algebraic characterizations of tree-width and clique-width, two graph complexity measures that are important for the construction of polynomial-time graph algorithms. Separately the properties of graphs can be studied in a logical language called monadic second-order logic. In this book, these two features of graph structure are brought together for the first time in a presentation that unifies and synthesizes research over the last 25 years. The authors not only provide a thorough description of the theory, but also detail its applications, on the one hand to the construction of graph algorithms, and on the other to the extension of formal language theory to finite graphs. This extension combines algebraic notions (equational and recognizable sets) and logical ones (graph transformations specified by logical formulas). Applications of these tools to languages of words and terms are also presented.

Consequently the book will be of interest to graduate students and researchers in graph theory, finite model theory, formal language theory and complexity theory.

**Encyclopedia of Mathematics and Its Applications**

This series is devoted to significant topics or themes that have wide application in mathematics or mathematical science and for which a detailed development of the abstract theory is less important than a thorough and concrete exploration of the implications and applications.

Books in the **Encyclopedia of Mathematics and Its Applications** cover their subjects comprehensively. Less important results may be summarized as exercises at the ends of chapters. For technicalities, readers can be referred to the bibliography, which is expected to be comprehensive. As a result, volumes are encyclopedic references or manageable guides to major subjects.

All the titles listed below can be obtained from good booksellers or from Cambridge University Press.
For a complete series listing visit www.cambridge.org/mathematics.

94 S. R. Finch *Mathematical Constants*
95 Y. Jabri *The Mountain Pass Theorem*
96 G. Gasper and M. Rahman *Basic Hypergeometric Series, 2nd edn*
97 M. C. Pedicchio and W. Tholen (eds.) *Categorical Foundations*
98 M. E. H. Ismail *Classical and Quantum Orthogonal Polynomials in One Variable*
99 T. Mora *Solving Polynomial Equation Systems II*
100 E. Olivieri and M. Eulália Vares *Large Deviations and Metastability*
101 A. Kushner, V. Lychagin and V. Rubtsov *Contact Geometry and Nonlinear Differential Equations*
102 L. W. Beineke and R. J. Wilson (eds.) with P. J. Cameron *Topics in Algebraic Graph Theory*
103 O. J. Staffans *Well-Posed Linear Systems*
104 J. M. Lewis, S. Lakshmivarahan and S. K. Dhall *Dynamic Data Assimilation*
105 M. Lothaire *Applied Combinatorics on Words*
106 A. Markoe *Analytic Tomography*
107 P. A. Martin *Multiple Scattering*
108 R. A. Brualdi *Combinatorial Matrix Classes*
109 J. M. Borwein and J. D. Vanderwerff *Convex Functions*
110 M.-J. Lai and L. L. Schumaker *Spline Functions on Triangulations*
111 R. T. Curtis *Symmetric Generation of Groups*
112 H. Salzmann *et al. The Classical Fields*
113 S. Peszat and J. Zabczyk *Stochastic Partial Differential Equations with Lévy Noise*
114 J. Beck *Combinatorial Games*
115 L. Barreira and Y. Pesin *Nonuniform Hyperbolicity*
116 D. Z. Arov and H. Dym *J-Contractive Matrix Valued Functions and Related Topics*
117 R. Glowinski, J.-L. Lions and J. He *Exact and Approximate Controllability for Distributed Parameter Systems*
118 A. A. Borovkov and K. A. Borovkov *Asymptotic Analysis of Random Walks*
119 M. Deza and M. Dutour Sikirić *Geometry of Chemical Graphs*
120 T. Nishiura *Absolute Measurable Spaces*
121 M. Prest *Purity, Spectra and Localisation*
122 S. Khrushchev *Orthogonal Polynomials and Continued Fractions*
123 H. Nagamochi and T. Ibaraki *Algorithmic Aspects of Graph Connectivity*
124 F. W. King *Hilbert Transforms I*
125 F. W. King *Hilbert Transforms II*
126 O. Calin and D.-C. Chang *Sub-Riemannian Geometry*
127 M. Grabisch *et al. Aggregation Functions*
128 L. W. Beineke and R. J. Wilson (eds.) with J. L. Gross and T. W. Tucker *Topics in Topological Graph Theory*
129 J. Berstel, D. Perrin and C. Reutenauer *Codes and Automata*
130 T. G. Faticoni *Modules over Endomorphism Rings*
131 H. Morimoto *Stochastic Control and Mathematical Modeling*
132 G. Schmidt *Relational Mathematics*
133 P. Kornerup and D. W. Matula *Finite Precision Number Systems and Arithmetic*
134 Y. Crama and P. L. Hammer (eds.) *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*
135 V. Berthé and M. Rigo (eds.) *Combinatorics, Automata and Number Theory*
136 A. Kristály, V. D. Rădulescu and C. Varga *Variational Principles in Mathematical Physics, Geometry, and Economics*
137 J. Berstel and C. Reutenauer *Noncommutative Rational Series with Applications*
138 B. Courcelle and J. Engelfriet *Graph Structure and Monadic Second-Order Logic*
139 M. Fiedler *Matrices and Graphs in Geometry*
140 N. Vakil *Real Analysis through Modern Infinitesimals*
141 R. B. Paris *Hadamard Expansions and Hyperasymptotic Evaluation*
142 Y. Crama and P. L. Hammer *Boolean Functions*
143 A. Arapostathis, V. S. Borkar and M. K. Ghosh *Ergodic Control of Diffusion Processes*
144 N. Caspard, B. Leclerc and B. Monjardet *Finite Ordered Sets*
145 D. Z. Arov and H. Dym *Bitangential Direct and Inverse Problems for Systems of Integral and Differential Equations*
146 G. Dassios *Ellipsoidal Harmonics*

# Foreword
# by Maurice Nivat

The genesis of this great and beautiful book spans more than 20 years. It collects and unifies many theoretical notions and results published by Bruno Courcelle and others in a large number of articles.

The concept of a language to communicate with a computer, a machine or any kind of device performing operations is at the heart of Computer Science, a field that has truly thrived with the emergence of symbolic programming languages in the 1960s. Formalizing the algorithms that enable computers to calculate an intended result, to control a machine or a robot, to search and find the relevant information in response to a query, and even to imitate the human brain in actions such as measuring risk and making decisions, is the main activity of computer scientists as well as of ordinary computer users.

The languages designed for these tasks, which number by thousands, are defined in the first place by syntactic rules that construct sets of words and to which are then attached meanings. This understanding of a language was first conceived by structural linguists, in particular Nicolaï Troubetskoï, Roman Jacobson and Noam Chomsky, and has transformed Linguistics, the study of natural languages, by giving it new directions. It has also been extended to programming languages, which are artificial languages, and to the Lambda Calculus, one of many languages devised by logicians, among whom we can cite Kurt Gödel, Alonzo Church and Alan Turing, who aspired to standardize mathematical notation and to mechanize proofs. This same idea has inspired all research on computation theory and programming. Thanks to the results of this research, planes can fly with continuously monitored flight parameters, providing us with unprecedented reliability: this is so because millions of lines of code have been formally proved to be correct.

Words are strings of symbols taken from finite alphabets. They constitute the basic elements. They can represent all the information one might wish to capture, use, process, disseminate or share in a world that is fast becoming more and more "digital," as Gérard Berry emphasized recently in his lectures at the Collège de France.

Most information, though represented always by words, is nevertheless structured hierarchically and can thus be presented in a natural way as a tree or as a graph. Most

of the countless electronic chips that make up computers but are also used in an ever-growing number of other machines as well, from washing machines to nuclear power plants, calculate on graphs: by connecting vertices, their edges can represent virtually any relationship of subordination, analogy, neighborhood or causality. From the early 1960s, the algorithms for graphs (and for trees, which are particular graphs) have been developed swiftly, and most of the current computing applications are based on these algorithms. Thousands of them have been designed by numerous researchers and engineers, and they fuel a burgeoning literature.

It was around 1980 that Bruno Courcelle, a former student of the prestigious Ecole Normale Supérieure (Rue d'Ulm in Paris), a logician by training, and a young but already established researcher, tackled a seemingly impossible task: to build a theory of tree languages that would classify all of these algorithms and present them in a unified and rational way. Bruno Courcelle is not a "problem solver" who happened to discover more-or-less elegant and clever answers to questions; he likes well-founded and harmonious theories, and is always looking for unifying concepts. Armed with a knowledge of logic and with a familiarity with Fundamental Computer Science, and in particular with Formal Language Theory which he gained during his years as a researcher at INRIA (Institut National de Recherche en Informatique et Automatique) while preparing his thesis, Bruno Courcelle got down to work with perseverance and determination.

Upon his arrival at Bordeaux-1 University in 1979 (LaBRI, the Laboratoire Bordelais de Recherche en Informatique, was created in 1988), Bruno Courcelle found an excellent work environment. The concept of attribute grammar, which is important in compilation, provided the model that he has used to develop an algebraic approach to graph grammars and a logical approach to the proof of properties of the graphs they generate. The first published work he devoted to attribute grammars is the source of the theory presented here, based on Logic and Universal Algebra.

The impact of his work surpassed all expectations, even taking into consideration the remarkable qualities of method and rigor that characterize Bruno Courcelle. For when the first elements of his theory began to spread among those who work on designing and improving graph algorithms, these researchers realized that Bruno Courcelle had provided a convenient formal framework in which many problems could be solved. In particular, Bruno's theory brought a logical lightening to the profound works of Paul Seymour and his collaborators on graph minors. Other researchers have been inspired by his theory to study new problems and invent new algorithms. A daunting theory that was originally seen as arcane and abstract proved to be rich and fertile. In 2004, Bruno Courcelle was awarded by INIST (an institute depending on the Centre National pour la Recherche Scientifique) and the ISI-Web of Science (Thomson-Reuters) the prize of the "most cited researcher in Computer Science in France."

I am not going to analyze his work further; in any case Chapter 1 is a long overview that is perfectly readable, even by those who are well versed neither in algebra nor in

mathematical logic. I would rather emphasize that the work of a computer scientist, as of any scientist, can be very diverse. The quest for new results is one endeavor, but bringing up to date the underlying structures, unifying concepts and simplifying the presentation of results, is quite another. The rapid, sometimes frantic, growth of publications in Computer Science has led most researchers to choose the former in pursuit of better results and more efficient algorithms. It gives me great pleasure to preface a work that is of the opposite nature: a long book produced by a comprehensive study, which leads to a very interesting result: the formation of a theory that brings order, that explains and simplifies a vast collection of results obtained by others and, at the same time, that proposes methods, yields results and raises new questions.

While still a student I was very struck when I first read André Lichnérowicz's book on linear algebra. I had already taken courses in linear algebra which, I confess, were not very helpful, and suddenly this book made everything clear. The mysterious operations which we were taught to perform on the square tables called "determinants" started making sense; the concepts of both vector space and the dimension of a vector subspace finally allowed me to understand what it meant to agonize over a determinant and, moreover, why this notion is important. Lichnérowicz's book is a classic that has enabled generations of students to learn linear algebra with ease, and it has become a mathematical tool widely used by engineers and technicians who are not professional mathematicians. I believe that this book will get the same reputation, quickly become a classic and provide an easy access to the burgeoning world of graph algorithms and its numerous applications throughout the sciences and beyond.

The comments above were written two years ago, when Bruno Courcelle's book was only 500 pages long, and I cannot change what I wrote then: it is a great and beautiful book that is going to take its place very soon on the library shelves of all the departments of Computer Science around the world. But now the book is 700 pages long and has two authors, Bruno and Joost Engelfriet. What happened is that Bruno sent the previous version to Joost to read and suggest corrections and improvements. Joost is a very old acquaintance of both Bruno and myself, and we have always known him as one of the most knowledgeable researchers in the field of grammars, automata and transducers on words and trees. And Joost had so many things to suggest that it is another book that I present today: thicker, with new results and a number of proofs that have been replaced by simpler and more elegant ones. Obviously the cooperation between Bruno and Joost was a very fruitful one indeed.

Knowing Joost as I do, this is not a surprise: when I asked him to referee papers submitted to the journal *Theoretical Computer Science*, in most cases Joost's report was longer and sometimes richer than the refereed article. His comments always led to a major improvement of the original text. Clearly Bruno's manuscript inspired Joost. And we all have to be grateful to him for, as usual, his comments and the work he did on the manuscript resulted in a major improvement and a sizable enlargement.

Thus today I am very happy to thank the two authors of this beautiful book, which I consider to be a wonderful source of knowledge in Computer Science. It is a theoretical

book, and for that reason some people may find it hard to read, but reading it is worth the pain, because the formalism introduced and the methods presented have already led to many new algorithms on graphs (as the number of citations of Bruno's published papers show) and they will lead to many others in the future. To anyone interested in graph algorithms I can only recommend that they read this book first.

For indeed this book lies at the very heart of Computer Science, which is the expressiveness of the languages used to represent and manipulate information and information structures, graphs being among the most widely used information structures. Progress in the efficiency, liability and simplicity of algorithms comes mainly from the use of better representations, better structures and a better understanding of the different ways in which one can describe sets of data and express their properties. This book provides a huge number of conceptual tools to design and study graph algorithms that no one should ignore.

In the name of the young but fast-growing science that in French we call *Informatics*, in the name of all future researchers in this field, I just say to Bruno and Joost: Thanks, you have done a good job!

# Contents

# Introduction

This book contributes to several fields of Fundamental Computer Science. It extends to finite graphs several central concepts and results of Formal Language Theory and it establishes their relationship to results about Fixed-Parameter Tractability. These developments and results have applications in Structural Graph Theory. They make an essential use of logic for expressing graph problems in a formal way and for specifying graph classes and graph transformations. We will start by giving the historical background to these contributions.

## Formal Language Theory

This theory has been developed with different motivations. Linguistics and compilation have been among the first ones, around 1960. In view of the applications to these fields, different types of *grammars*, *automata* and *transducers* have been defined to specify *formal languages*, i.e., sets of *words*, and transformations of words called *transductions*, in finitary ways. The formalization of the semantics of sequential and parallel programming languages, that uses respectively *program schemes* and *traces*,[1] the modeling of biological development and yet other applications have motivated the study of new objects, in particular of sets of *terms*.[2] These objects and their specifying devices have since been investigated from a mathematical point of view, independently of immediate applications. However, all these investigations have been guided by three main types of questions: comparison of descriptive power, closure properties (with effective constructions in case of positive answers) and decidability problems.

A *context-free grammar* generates words, hence specifies a formal language. However, each generated word has a *derivation tree* that represents its structure relative to the considered grammar. Such a tree, which can also be viewed as a term, is usually

[1] Traces are equivalence classes of words for congruences generated by commutations of letters; see the book [*DiekRoz]. For program schemes, see [*Cou90a]. The list of references is divided into two parts. The first part lists books, book chapters and survey articles: the * in, e.g., [*DiekRoz] indicates a reference of this kind. The second part lists research articles and dissertations.

[2] In Semantics, one is also interested in *infinite* words, traces and terms. In this book these will not be considered.

the support of further computation, typically a translation into a word of another language (this is the case in linguistics and in compilation). Hence, even for its initial applications, Formal Language Theory has had to deal with trees as well as with words. In Semantics, terms are even more important than words. Thus, sets of terms, usually called *tree languages*[3], and transductions of terms, called *tree transductions*, have become central notions in Formal Language Theory.

Together with context-free grammars, *finite* (also called *finite-state*) *automata* are among the basic notions of Language Theory, in particular for their applications to lexical analysis and pattern matching. They were also used early on (around 1960) for building algorithms to check the validity of certain logical formulas, especially those of *monadic second-order logic*, in certain relational structures. On the other hand, monadic second-order logic can be used to specify and to classify sets of words and terms.[4] There are deep relationships between monadic second-order formulas and finite automata that recognize words and terms (see [*Tho97a]). The *fundamental result* is that every language that is specified by a sentence of monadic second-order logic (expressing a property of words) can be recognized by a finite automaton, and vice-versa. Moreover, the finite automaton can be constructed effectively from the sentence. This means that monadic second-order logic can be viewed as a high-level specification language that can be compiled into "machine code": a finite automaton that recognizes the words that satisfy the specification. The same result holds for terms, with respect to finite automata on trees. As a consequence of this fundamental relationship, monadic second-order logic is now one of the basic tools used in Formal Language Theory and its applications, in addition to context-free grammars, finite automata and finite transducers (which are finite automata with output).

The extension of the basic concepts of Formal Language Theory to *graphs* is a natural step because graphs generalize trees. However, graphs have already been present from the beginnings in several of its fields. In compilation, one uses *attribute grammars* that are context-free grammars equipped with semantic rules ([*AhoLSU], [*Cre]). These rules associate graphs (called *dependency graphs*) with derivation trees. An attribute grammar is actually the paradigmatic example of a *context-free graph grammar* (based on *hyperedge replacement* rewriting rules, [*DreKH]). In the semantics of parallelism, traces are canonically represented by graphs, and an important concern is to specify them by finite automata ([*DiekRoz]).

One starting point of the research presented in this book has been the development of a robust theory of *context-free graph grammars*, of *recognizability of sets of graphs* (to be short, an algebraic formulation of finite automata) and of *graph transductions*. In order to use the theory of context-free grammars and recognizability in arbitrary algebras initiated by Mezei and Wright in [MezWri], we choose appropriate

[3] In addition to being words, terms have canonical representations as labeled, rooted and ordered trees. They are thus called "trees" but this terminology is inadequate.

[4] This logical language and the related one called $\mu$-calculus ([*ArnNiw]) are also convenient for expressing properties of programs.

(and natural) operations on graphs. Thus, graphs become the value of terms that are built with these (infinitely many) operations. Roughly speaking, a *context-free graph grammar* is a finite set of rules of the form $A_0 \to f(A_1,\ldots,A_n)$, $n \geq 0$, where each $A_i$ is a nonterminal of the grammar and $f$ is one of the chosen graph operations. The rule means that if the graphs $G_1,\ldots,G_n$ are generated by respectively $A_1,\ldots,A_n$, then $A_0$ can generate the graph $f(G_1,\ldots,G_n)$. Such grammars have useful applications to Graph Theory: they can be used to describe many graph classes in uniform ways and to prove by inductive arguments certain properties of their graphs. Still roughly speaking, a set of graphs is *recognizable* if there is a finite automaton that recognizes all the terms that evaluate to a graph in the set. Thus, the automaton does not work directly on the given graph, but rather on any term that represents that graph. In a similar way one can define graph transductions through the use of tree transducers. Note that, to describe a set of graphs or a graph transduction in a finitary way, one can necessarily use only finitely many graph operations. As we will see, that is a rather severe, but natural restriction.

Our main goal will be to show that the fundamental use of monadic second-order logic as a high-level specification language carries over to graphs, not only for the specification of recognizable sets of graphs, but also for context-free sets of graphs and for certain types of graph transductions. This gives a new dimension to the above-mentioned fundamental result for words and terms, because the properties of graphs that can be specified in monadic second-order logic are more varied and useful than those of words and terms.

We will specify a set of graphs by a monadic second-order sentence, and a graph transduction by a tuple of monadic second-order formulas that define an "interpretation" of the output graph in the input graph. From such a specification we will show how one can construct a finite automaton on terms, or a tree transducer in the second case, that is related to the specification as explained above. Note that the logic "acts" directly on the graphs, whereas the automata and transducers work on the terms that denote these graphs. Thus, monadic second-order logic can be viewed as playing the role of "finite automata on graphs" and "finite transducers of graphs" in our Formal Language Theory for Graphs.

### Graph algorithms

The above-mentioned developments have important applications for the construction of polynomial-time algorithms on graphs. In his 16th NP-completeness column, published in 1985 [John], Johnson reviews a number of NP-complete graph problems that become polynomial-time solvable if their inputs are restricted to particular classes of graphs such as those of trees, of series-parallel graphs, of planar graphs to name a few. For many of these classes, in particular for trees, *almost trees* (with parameter $k$), *partial k-trees*, *series-parallel graphs*, *outerplanar graphs* and *cographs*, the efficient algorithms take advantage of certain hierarchical structures of the input

graphs. Because of these structures, these graphs are somehow close to trees.[5] The notion of a partial $k$-tree has emerged as a powerful one subsuming many other types of "tree-like graphs." (The cographs have a canonical hierarchical structure but they are not included in the class of partial $k$-trees for any fixed $k$.) Many articles have produced polynomial-time algorithms for NP-complete problems restricted to partial $k$-trees. In 1994, Hedetniemi has compiled a list of 238 references [*Hed] on partial $k$-trees and algorithms concerning them. The notion of a partial $k$-tree has also been used with a different terminology (*tree-width*, *tree-decomposition*) by Robertson and Seymour in their study of the structure of graph classes that exclude fixed graphs as minors. They formulate this notion in terms of particular decompositions of graphs, called tree-decompositions, that are at the basis of the construction of polynomial-time algorithms. Each tree-decomposition has a *width*, and a graph is a partial $k$-tree if and only if it has tree-width at most $k$, which means that it has a tree-decomposition of width at most $k$.

The recent theory of Fixed-Parameter Tractability (the founding book by Downey and Fellows [*DowFel] was published in 1999) now gives a conceptual framework to most of these results. The notion of a *fixed-parameter tractable algorithm* specifies how the multiplicative constant factor of the time-complexity of a polynomial-time algorithm depends on certain parts of the data. It happens that for most of the graph algorithms based on tree-decompositions, the exponent of the polynomial is 1: these algorithms are linear-time in the size of the input graphs, with multiplicative "constant" factors that depend exponentially (or more) on the widths of the input tree-decompositions.

The explanation for this fact is one of the main goals of this book. We will show that, for a certain natural choice of graph operations, tree-decompositions correspond to terms, and tree-decompositions of width at most $k$ correspond to terms that are built from a finite subset of those operations. A general algorithmic result that encompasses many of the above-mentioned results, follows from the fundamental relationship between monadic second-order logic and finite automata discussed before: if the considered problem is specified by a monadic second-order sentence (and this is the case for many NP-complete graph problems not using numerical values in their inputs), then a finite automaton on the terms that encode the tree-decompositions of width at most $k$ can be constructed (for each $k$) to give the answer to the considered question (for example, *Is the given graph 3-colorable?*) where the input graph is given by a tree-decomposition (or a term encoding it). The linearity result follows because finite automata can be implemented so as to work in linear time (and because a tree-decomposition of a graph can be found in linear time).

[5] These classes can actually be generated by certain context-free graph grammars and the corresponding hierarchical structures of the generated graphs are represented by their derivation trees. There is thus a close relationship between the algorithmic issues and the extensions of language theoretic concepts discussed above.

We will extend the case of tree-width bounded graphs (already discussed in [*DowFel]) to another type of graph decompositions, based on another natural choice of graph operations. This leads to the notion of *clique-width* of a graph. Clique-width is more powerful than tree-width in the sense that every set of graphs of bounded tree-width has bounded clique-width but not vice-versa, an example being the set of cographs. On the other hand, in the above general result, the monadic second-order sentences must be restricted to use quantifications on sets of vertices (instead of both vertices and edges), so fewer graph problems can be specified. The algorithms are cubic-time instead of linear-time because, for these graph operations, cubic time is needed to find a term for a given graph.

The theory that will be exposed in the nine chapters of this book has arisen from the confluence of the two main research directions presented above. The remainder of this introduction will present in a more detailed way, but still informally, the main concepts and results.

## The role of logic

We will study and compare finitary descriptions of sets of finite graphs by using concepts from Logic, Universal Algebra and Formal Language Theory. We first explain the role of Logic. A graph[6] can be considered as a *logical structure* (also called *relational structure*) whose *domain* (also called its *universe*) consists of the vertices, and that is equipped with a binary relation that represents adjacency. Graph properties can thus be expressed by logical formulas of different languages and classified accordingly.

*First-order* formulas are rather weak in this respect because they can only express local properties such as that a graph has maximum degree or diameter bounded by a fixed integer. Most properties of interest in Graph Theory can be expressed by *second-order formulas*: these formulas can use quantifications on relations of arbitrary arity. Unfortunately, little can be obtained from the expression of a graph property in second-order logic. Our favorite logical language will be its restriction called *monadic second-order logic*. Its formulas are the second-order formulas that only use quantifications on unary relations, i.e., on sets. They can express many useful graph properties like *connectivity*, *p-colorability* (for fixed $p$) and *minor inclusion*, whence *planarity*. Such properties are said to be *monadic second-order expressible*, and the corresponding sets of graphs are *monadic second-order definable*.

These logical expressions have interesting algorithmic consequences as explained above, but only for graphs that are somehow "tree-like" (because 3-colorability is NP-complete and expressible by a monadic second-order sentence). Monadic second-order sentences are also used in Formal Language Theory to specify *languages*, i.e., sets of words or terms. The fundamental result establishes that monadic second-order sentences and finite automata have the same descriptive power. But

[6] In order to simplify the discussion, we only discuss simple graphs, i.e., graphs without parallel edges.