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 μ-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 $A_0 \to f(A_1,\dots,A_n)$, $n \geq 0$, 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 $G_1,\dots,G_n$ are generated by respectively $A_1,\dots,A_n$, then A_0 can generate the graph $f(G_1,\dots,G_n)$. 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.

monadic second-order formulas are even more important for specifying sets of graphs than for specifying languages because there is no convenient notion of graph automaton. They replace finite automata, not only for specifying sets of graphs, but also for specifying graph transformations. Such transformations, called *monadic second-order transductions*, generalize the transductions of terms and words defined by finite automata with output called *finite transducers*.[7] Independently of these language theoretic applications, monadic second-order transductions are technically useful for constructing monadic second-order formulas because the inverse image of a monadic second-order definable set of relational structures under a monadic second-order transduction is monadic second-order definable.

However, monadic second-order logic alone yields no interesting results. In order to be useful for the construction of algorithms, the expression of a graph property by a monadic second-order sentence must be coupled with constraints on the graphs of interest such as having bounded tree-width or bounded clique-width. The language theoretical issues to be discussed below will also combine monadic second-order sentences and the very same constraints. Hence, we will study certain *hierarchical graph decompositions*, such as tree-decompositions, that fit with monadic second-order logic.

Graph algebras

Graph decompositions will be formalized algebraically by terms written with appropriate graph operations. Hence, we will use concepts from Universal Algebra in addition to ones from Logic.

For treating graphs as algebraic objects, i.e., as elements of appropriate algebras (words and traces are elements of monoids), we will define *graph operations* that generalize the concatenation of words. We will consider two natural ways to "concatenate" two graphs. One way is to "glue" them together, by identifying some of their vertices. The other way is to "bridge" them (or rather, "bridge the gap between them"), by adding edges between their vertices. Clearly, to obtain single valued operations, we have to specify which vertices must be "glued" or "bridged." By means of labels attached to vertices, we will specify that vertices with the same label must be identified, or that edges must be created between all vertices with certain labels. Hence, we will define "concatenation" operations on *labeled graphs*. To allow the flexible use of vertex labels, we also define (unary) operations that modify these labels. Terms written with these operations evaluate to finite (labeled) graphs. The value G of a term $t = f(t_1, t_2)$ is a certain combination, specified by f, of the values of its subterms t_1 and t_2. These values are, roughly speaking, subgraphs of G (only "roughly" because the labels of the vertices of the graphs defined by t_1 and t_2 may differ from their labels in the resulting graph G). The same holds for all subterms of t, hence, t represents a hierarchical decomposition of G.

[7] In particular, the *rational transductions* that are transductions of words defined either by finite(-state) transducers or, algebraically, in terms of homomorphisms and regular languages.

Based on the idea of "gluing" graphs (and using the numbers $1,\ldots,k+1$ as labels), we will define, for each k, a finite set of graph operations, $F^{\mathrm{HR}}_{[k+1]}$, that generates exactly the graphs of tree-width at most k. Hence, these operations formalize algebraically an existing combinatorial notion. They yield a graph algebra (that generalizes the monoid of words) having countably many operations. We will call it the *HR algebra* for reasons explained below. Another countable family of graph operations, also indexed by positive integers and based on the idea of "bridging" graphs, will yield a different graph algebra, called the *VR algebra*, and a graph complexity measure called *clique-width*. By definition, a graph has clique-width at most k if it is generated by the analogous finite set of graph operations $F^{\mathrm{VR}}_{[k]}$. As observed before, 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. Many definitions and results will be similar for these two graph algebras. We will explain below why both algebras are interesting.

The introduction of graph operations is essential for our project of extending to graphs the basic concepts of Formal Language Theory in a clean way. We will use for that the algebraic notions of an *equational* set and of a *recognizable* set. An equational set is a component of the least solution of an equation system written with set union and the operations of the considered algebra. Equation systems formalize context-free grammars that generate elements of the algebra: if such a context-free grammar has, e.g., three rules $A \to f(B,C)$, $A \to g(A)$ and $A \to a$ for the nonterminal A, where B and C are two other nonterminals, f and g are operations of the algebra and a is a constant of the algebra, then the corresponding equation system has the equation $A = f(B,C) \cup g(A) \cup \{a\}$ (where A, B and C now stand for sets of elements of the algebra). The context-free languages are actually the equational sets of the monoids of words over their terminal alphabets (due to the least fixed-point characterization of context-free grammars of [GinRic] and [ChoSch]). A recognizable set is a set saturated by a congruence having finitely many classes. The regular languages are thus the recognizable sets of the monoids of words. When all elements of the algebra can be denoted by a term (which is the case for the HR and VR algebra), a set is recognizable if and only if there exists a finite automaton on terms that recognizes all the terms that evaluate to an element of the set.

The chart of Figure I.1 shows some relationships between the notions defined above. An arrow means "used for a definition or a construction."

Two graph algebras

Since we will define two graph algebras, we will obtain two types of equational sets, called the HR- and the VR-equational sets. For each k, the set of graphs of tree-width at most k is HR-equational (because it is generated by the finite set of operations $F^{\mathrm{HR}}_{[k+1]}$), and similarly, the set of graphs of clique-width at most k is VR-equational. There are also two types of recognizable sets of graphs, the HR- and the VR-recognizable sets. Every HR-equational set is VR-equational and every VR-recognizable set is

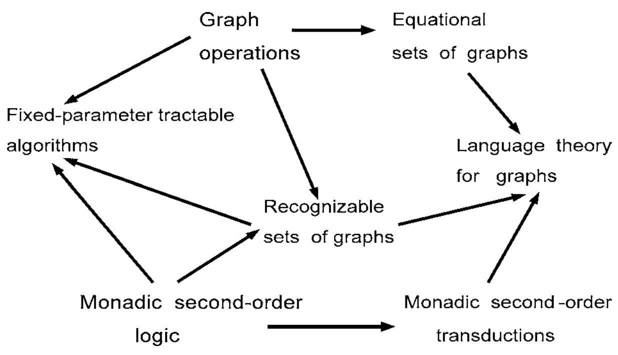
Graph operations
Equational sets of graphs
Fixed-parameter tractable algorithms
Language theory for graphs
Recognizable sets of graphs
Monadic second-order logic
Monadic second-order transductions

Figure I.1 The main notions.

HR-recognizable, but not vice-versa. The class of HR-equational sets is incomparable with the class of HR-recognizable sets, and similarly for the VR algebra.

These facts show some important differences with the case of words. For words, we have a unique algebraic structure based on a single operation, and the class of recognizable sets (the regular languages) is properly included in that of equational sets (the context-free languages). But graphs are intrinsically more complicated than words: this explains why we need countably many operations and not just one. We will explain next why we have *two* algebras and two (robust) classes of equational sets that both generalize the class of context-free languages.

The two graph algebras have been defined initially in such a way that their equational sets coincide with existing *context-free sets of graphs*: the HR-equational sets are actually (but not by definition) those generated by certain context-free graph grammars based on a rewriting mechanism called *hyperedge replacement* (that uses "gluing" of graphs) and we call the corresponding algebra the HR algebra to refer to this fact; the other algebra, called the VR algebra, has been designed similarly so that its equational sets are those generated by the context-free graph grammars based on *vertex replacement* (that uses "bridging" of graphs); see [*DreKH] and [*EngRoz] respectively for these two types of graph grammars.

Many properties of the equational and recognizable sets of graphs of both kinds are just particular instances of those of the equational and recognizable sets in arbitrary algebras. By using this algebraic approach, we generalize the context-free languages without having to define a graph rewriting mechanism and check that such rewriting is actually context-free (the general notion of context-free rewriting is defined in [Cou87]). Similarly, we generalize the regular languages without having to define any notion of graph automaton and to look for the closure properties of the class of sets of graphs that are recognized by such automata.

Monadic second-order logic and the VR graph algebra

We first discuss the equational sets and the recognizable sets of the VR algebra, and their relationships with monadic second-order logic.

Two main results of this book are the Recognizability Theorem and the Equationality Theorem. They relate two ways of handling graphs: the "logical way" by which graphs are characterized in terms of what they are made of and contain (vertices, edges, paths, minors, subgraphs with particular properties) and the "algebraic way" by which sets of graphs are characterized more globally by means of equation systems and congruences. In the latter approach, graphs are treated as elements of algebras and related with other elements that are not necessarily among their subgraphs.

The *Recognizability Theorem* says that if a set of graphs is monadic second-order definable, then it is VR-recognizable. The *Equationality Theorem* says that a set of graphs is VR-equational if and only if it is the image of the set of finite trees under a monadic second-order transduction[8]. We now describe some consequences of these two results.

The Recognizability Theorem entails that if a graph G is defined by a term t written with operations of the VR algebra belonging to any fixed finite set F, then one can check in time $O(|t|)$ whether or not G satisfies a fixed monadic second-order property. This fact, based on a compilation of monadic second-order formulas into finite automata over F, is one of the keys[9] to the construction of fixed-parameter tractable algorithms for the verification of monadic second-order properties of graphs of bounded clique-width (whence also of graphs of bounded tree-width since bounded tree-width implies bounded clique-width). Another consequence is the *Filtering Theorem*, which says that the graphs of a VR-equational set that satisfy a fixed monadic second-order property (for example planarity) form a VR-equational set. This is based on a Filtering Theorem that holds in all algebras and says that the intersection of an equational set and a recognizable one is equational (generalizing the corresponding fact for context-free and regular languages). Since the emptiness of an equational set is decidable, we get as another corollary that the *monadic second-order satisfiability problem* is decidable for each VR-equational set L. This means that one can decide whether or not a given monadic second-order sentence is satisfied by some graph in L.

The Equationality Theorem entails that the class of VR-equational sets of graphs is preserved under monadic second-order transductions, because the class of monadic second-order transductions is closed under composition. This corollary strengthens the Filtering Theorem. It is similar to the fact that the image of a context-free language under a rational transduction is context-free.

Monadic second-order logic and the HR graph algebra

The Recognizability and Equationality Theorems have versions relative to the HR algebra. To describe them, we must go back to the initial definition of monadic second-order formulas (MS formulas in the sequel) interpreted in graphs: they only

[8] This means, informally, that it is the set of graphs "defined inside finite trees" by a fixed finite tuple of monadic second-order formulas. These transductions are based on, and extend, the model-theoretical notion of "interpretation."
[9] The other one is a polynomial-time algorithm that finds a term evaluating to a given graph G if one exists.

use quantifications on vertices and sets of vertices. This is due to the chosen representation of a graph by a relational structure whose domain is its set of vertices. However, we can also express logically the properties of a graph G via its *incidence graph* $Inc(G)$. The vertices of this (bipartite) graph are the vertices and the edges of G, and its adjacency relation links a vertex and the edges incident with it. Thus, monadic second-order formulas to be interpreted in $Inc(G)$ (MS_2 formulas in the sequel) can also use quantifications on edges and sets of edges. A graph property is MS_2-*expressible* if it is expressible by an MS_2 formula, and the corresponding set of graphs is MS_2-*definable*. The notation MS_2 refers to this extension of the initially defined language (referred to as MS in the sequel). It is strictly more expressive. For example, the existence of a perfect matching is MS_2-expressible but not MS-expressible. However, MS_2 formulas are not more expressive than MS formulas for properties of words, of trees and of certain types of graphs such as planar graphs and, for each k, of graphs of degree at most k. These facts show the existence of deep links between structural graph properties (such as planarity) and the expressive power of MS_2 versus MS sentences.

The Recognizability Theorem for the HR algebra says that every MS_2-definable set of graphs is HR-recognizable, and the Equationality Theorem says that a set of graphs is HR-equational if and only if the set of its incidence graphs is the image of the set of finite trees under a monadic second-order transduction. We obtain an algorithmic consequence similar to the one we have discussed for MS-expressible problems and the VR algebra: if a graph is defined by a term t over $F_{[k]}^{\mathrm{HR}}$ for some fixed k, then one can check in time $O(|t|)$ whether or not it satisfies a fixed MS_2 property. Since there exists a polynomial-time algorithm that decomposes appropriately the input graphs, we obtain, for each MS_2 property, a fixed-parameter tractable verification algorithm, tree-width being the parameter. The algorithm for MS properties applies to larger classes of graphs, because bounded tree-width implies bounded clique-width, but to less properties than this one, because not every MS_2-expressible property is MS-expressible. The notions of tree-width and clique-width are thus both useful, for solving different problems. We also have a Filtering Theorem for the HR-equational sets and MS_2-expressible properties, whence the decidability of the MS_2-satisfiability problem for each HR-equational set. The Equationality Theorem for the HR algebra entails that the class of HR-equational sets of graphs is preserved under the monadic second-order transductions that transform incidence graphs.

A graph is uniformly k-sparse if its number of edges is at most k times its number of vertices, and the same holds for all its subgraphs. Another main result of this book is the *Sparseness Theorem*: MS_2 formulas are not more expressive than MS formulas for properties of uniformly k-sparse graphs, for each fixed k. The above-mentioned types of graphs are uniformly k-sparse for some k.