Bruno Courcelle and Joost Engelfriet

Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach

April 2011

to be published by

Cambridge University Press

Chapter 1

Overview

This chapter presents the main definitions and results of this book and their significance, with the help of a few basic examples. It is written so as to be readable independently of the others. Definitions are sometimes given informally, with simplified notation, and most proofs are omitted. All definitions will be repeated with the necessary technical details in the subsequent chapters.

In Section 1.1, we present the notion of equational set of an algebra by using as examples a context-free language, the set of cographs and the set of series-parallel graphs. We also introduce our algebraic definition of derivation trees.

In Section 1.2, we introduce the notion of recognizability in a concrete way, in terms of properties that can be proved or refuted, for every element of the considered algebra, by an induction on any term that defines this element. We formulate a concrete version of the Filtering Theorem saying that the intersection of an equational set and a recognizable one is equational. It follows that one can decide if a property belonging to a finite inductive set of properties is valid for every element of a given equational set. We explain the relationship between recognizability and finite automata on terms.

In Section 1.3, we show with several key examples how monadic second-order sentences can express graph properties. We recall the fundamental equivalence of monadic second-order sentences and finite automata for words and terms.

In Section 1.4, we introduce two graph algebras. They are called the VR and the HR algebra because their equational sets are those that are generated by the context-free Vertex Replacement and Hyperedge Replacement graph grammars respectively. The cographs and the series-parallel graphs are respectively our current examples of a VR- and an HR-equational set. We state (a weak version of) the Recognizability Theorem which says, in short, that monadic second-order definability implies recognizability. From it we obtain a logical version of the Filtering Theorem where the recognizable sets are defined by monadic second-order sentences.

In Section 1.5, we review the basic definitions of Fixed-Parameter Tractability and we state the algorithmic consequences of the (weak) Recognizability

Theorem. This theorem has actually two versions, relative to the two graph algebras defined in Section 1.4, and yields two Fixed-Parameter Tractability Theorems.

In Section 1.6, we describe the consequences of the Recognizability and Filtering Theorems for the problem of deciding whether a given monadic second-order sentence is satisfied by some graph of tree-width at most a given k or, more generally, by some graph of an equational set.

In Section 1.7, we introduce the notion of a monadic second-order transduction by means of examples that have some graph theoretic content, and we state the Equationality Theorem for the VR algebra. It gives a characterization of the VR-equational sets, and in particular of the sets of graphs of bounded clique-width, that is formulated in purely logical terms.

In Section 1.8, we consider monadic second-order formulas interpreted in incidence graphs (as opposed to in graphs "directly"). These formulas can use edge set quantifications. We compare the corresponding four types of monadic second-order transduction and we state the Equationality Theorem for the HR algebra: it is based on monadic second-order transductions that transform incidence graphs.

In Section 1.9, we define relational structures and we extend to them (easily) some results relative to graphs represented by their incidence graphs. We introduce betweenness and cyclic ordering as examples of combinatorial notions that are based on linear orderings but are defined in a natural way as ternary relations.

1.1 Context-free grammars

By starting from the standard notion of a context-free grammar, we introduce the notion of an equational set and we define two equational sets of graphs. We define the equational sets of a (one-sorted) algebra and the corresponding sets of derivation trees.

1.1.1 Context-free word grammars

By using *context-free grammars*, one can specify certain formal languages, namely the *context-free languages*, in a finitary way. Context-free grammars are usually defined as rewriting systems satisfying particular properties, conveyed by the term "context-free" and axiomatized in [Cou87]. However, the *Least Fixed-Point Characterization* of context-free languages due to Ginsburg and Rice [GinRic] and to Chomsky and Schützenberger [ChoSch] is formulated in terms of systems of recursive equations written with the operations of union and concatenation over languages. This algebraic view has been developed by Mezei and Wright [MezWri] and has many advantages. First, it is more synthetic in that it deals with languages rather than with words produced individually by derivation sequences. Second, it puts the study of context-free languages in the more general

framework of recursive definitions handled as least solutions of systems of equations, and last but not least, it is applicable to any algebra. This latter aspect is especially important for the extension to graphs.

We recall how context-free languages can be characterized as the components of the least solutions of certain systems of equations in languages. A context-free grammar G is a finite set of rewriting rules defined with two alphabets, a *terminal alphabet* A and a *nonterminal alphabet* N. For every S in N, the context-free language over A generated by G from S is denoted by $L(G, S)$.

Example 1.1 We consider for example the context-free grammar G with terminal alphabet $A = \{a, b, c\}$, nonterminal alphabet $N = \{S, T\}$ and rules named respectively $p, q, \ldots, w$ (where ε denotes the empty word):

$$\begin{array}{ll} p: & S \to aST \\ q: & S \to SS \\ r: & S \to a \\ s: & T \to bTST \\ u: & T \to a \\ v: & T \to c \\ w: & T \to \varepsilon \end{array}$$

It defines two languages $L(G, S)$ and $L(G, T)$ over A, i.e., sets of words in A^*. These languages satisfy the equations of the following system Σ_G:

$$\Sigma_G \left\{ \begin{array}{lcl} K & = & aKL \cup KK \cup \{a\} \\ L & = & bLKL \cup \{a, c, \varepsilon\} \end{array} \right.$$

with $K = L(G, S)$ and $L = L(G, T)$. The pair $(L(G, S), L(G, T))$ is thus a solution of Σ_G. However, it is not the only one. The pair of languages $(A^*, bA^* \cup \{a, c, \varepsilon\})$ is another solution as one checks easily[1]. The Least Fixed-Point Characterization of context-free languages establishes that the pair $(L(G, S), L(G, T))$ is the least solution of Σ_G for component-wise inclusion.

1.1.2 Cographs

We give two examples of similar definitions of sets of graphs. We first consider as ground set the set $\mathcal{G}^{\mathrm{u}}$ of undirected simple graphs[2]. Two isomorphic graphs are considered as the same object. We will use $\oplus$ to denote the disjoint union of two graphs G and H. This means that $G \oplus H$ is the union of G and of a copy of H disjoint with G (hence $G \oplus G \neq G$). We will also use the *complete join*, $G \otimes H$, defined as $G \oplus H$ augmented with undirected edges linking every vertex

[1] Since $L(G, S) \subseteq A^+ = AA^*$, the pair $(A^*, bA^* \cup \{a, c, \varepsilon\})$ is a solution of Σ_G that differs from $(L(G, S), L(G, T))$.

[2] In this book, all graphs are finite. A graph is *simple* if it has no two *parallel edges*, i.e., no two edges with the same ends, and the same directions in the case of directed graphs. Parallel edges are also called *multiple* edges. An edge with equal ends is a *loop*. The superscript "u" in $\mathcal{G}^{\mathrm{u}}$ refers to undirected graphs.

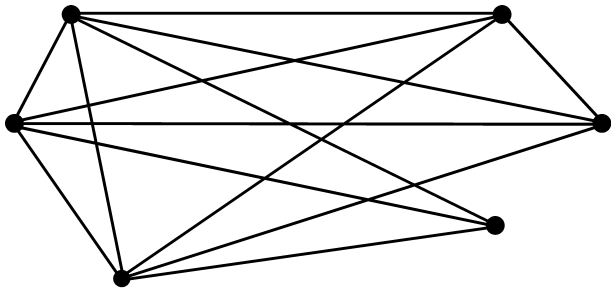

Figure 1.1: A cograph.

of G and every vertex of H. We let $\mathbf{1}$ denote any graph with one vertex and no edges. Note that both $\oplus$ and $\otimes$ are commutative and associative operations.

The *set of cographs* C can be defined as the least set of graphs satisfying the equation

$$C \quad = \quad (C \oplus C) \cup (C \otimes C) \cup \{\mathbf{1}\}. \tag{1.1}$$

This set (it is a proper subset of $\mathcal{G}^{\mathrm{u}}$) has actually alternative characterizations (see Section 1.3.1 below). From this equation, one can derive definitions of certain subsets of C. Consider for example the following system of two equations:

$$\left\{ \begin{array}{lcl} C_0 & = & (C_0 \oplus C_0) \cup (C_1 \oplus C_1) \cup (C_0 \otimes C_0) \cup (C_1 \otimes C_1) \\ C_1 & = & (C_0 \oplus C_1) \cup (C_0 \otimes C_1) \cup \{\mathbf{1}\}. \end{array} \right. \tag{1.2}$$

Its least solution in $\mathcal{P}(\mathcal{G}^{\mathrm{u}}) \times \mathcal{P}(\mathcal{G}^{\mathrm{u}})$ is the pair of sets (C_0, C_1) where C_0 (resp. C_1) is the set of cographs having an even (resp. an odd) number of vertices.[3] We will give general and effective methods for deriving from an equation or a system of equations that defines a set L, an equation or a system of equations defining $\{x \in L \mid P(x)\}$ where P is a property of the objects under consideration. This is possible if P has an appropriate "inductive behaviour" relative to the operations with which the given equation or system of equations is written.

From the definition of cographs as elements of the least subset C of $\mathcal{G}^{\mathrm{u}}$ satisfying (1.1), it follows that each of them is *denoted by a term*, more formally, is the *value of a term* in an *algebra* of graphs. Examples of terms denoting cographs are

$$\mathbf{1},\ \mathbf{1} \oplus \mathbf{1},\ (\mathbf{1} \oplus \mathbf{1}) \otimes \mathbf{1},\ (\mathbf{1} \oplus \mathbf{1}) \otimes (\mathbf{1} \oplus \mathbf{1}).$$

The cograph of Figure 1.1 is the value of the term $t = (\mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}) \otimes (\mathbf{1} \oplus (\mathbf{1} \otimes \mathbf{1}))$. Since $\otimes$ is associative, we have written t by omitting some parentheses as usual, for readability. These terms belong to the set $T(\{\oplus, \otimes, \mathbf{1}\})$ of all terms written with the constant $\mathbf{1}$ and the two binary operations $\oplus$ and $\otimes$. Equation (1.1) can also be solved with ground set $T(\{\oplus, \otimes, \mathbf{1}\})$. For this interpretation of (1.1) the unknown C denotes subsets of $T(\{\oplus, \otimes, \mathbf{1}\})$. Clearly, the set

[3] We denote by $\mathcal{P}(X)$ the powerset of a set X, i.e., its set of subsets.

of terms $T(\{\oplus, \otimes, \mathbf{1}\})$ itself is the least (in fact, the only) solution of (1.1) in $\mathcal{P}(T(\{\oplus, \otimes, \mathbf{1}\}))$.

A similar fact holds for System (1.2). Its least solution in $\mathcal{P}(T(\{\oplus, \otimes, \mathbf{1}\})) \times \mathcal{P}(T(\{\oplus, \otimes, \mathbf{1}\}))$ is a pair of sets (T_0, T_1) where $T_0,\ T_1 \subseteq T(\{\oplus, \otimes, \mathbf{1}\})$ and for each $i = 0, 1$, the set C_i is the set of cographs which are the values of the terms in T_i.

This example shows that a *grammar*, i.e., a system of equations like (1.2), specifies not only a tuple of sets of objects, here graphs, but also denotations by terms of the specified objects. These objects can be words, terms, trees, graphs, as we will see. Each term is a formalization of the structure of the object it denotes, as specified by the grammar; it provides a hierarchical decomposition of that object. In many cases, an object can be denoted by several terms that are correct with respect to the grammar. In such a case, we say that the grammar is *ambiguous*. The grammars (1.1) and (1.2) are ambiguous: since $\oplus$ and $\otimes$ are commutative and associative, most cographs are denoted by more than one term.

As an example of structure, consider again the term $(\mathbf{1}\otimes\mathbf{1}\otimes\mathbf{1})\otimes(\mathbf{1}\oplus(\mathbf{1}\otimes\mathbf{1}))$ that denotes the cograph of Figure 1.1. It provides a decomposition of that cograph, because the subterm $\mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}$ denotes the triangle at the left of Figure 1.1, whereas the subterm $\mathbf{1} \oplus (\mathbf{1} \otimes \mathbf{1})$ denotes the three vertices at the right of Figure 1.1 together with the edge between two of them.

1.1.3 Series-parallel graphs

The ground set of graphs is here the set $\mathcal{J}_2^{\mathrm{d}}$ of directed graphs G equipped with two distinct distinguished vertices marked 1 and 2 called its *sources*, denoted respectively by $src_G(1)$ and $src_G(2)$. These graphs may have multiple edges[4]. Let e be a constant denoting the graph with two vertices and only one edge from source 1 to source 2. The operations are the *parallel-composition*, denoted by $/\!\!/$, and the *series-composition*, denoted by $\bullet$. For G and H in $\mathcal{J}_2^{\mathrm{d}}$, the graph $G/\!\!/H$ is the union of G and an isomorphic copy H' of H such that $src_G(1) = src_{H'}(1)$, $src_G(2) = src_{H'}(2)$, and G and H' have nothing else in common. We define $src_{G/\!\!/H}(1) := src_G(1)$ and $src_{G/\!\!/H}(2) := src_G(2)$. Note that $G /\!\!/ G$ has twice as many edges as G, hence $G \neq G /\!\!/ G$ in general.

Series-composition is defined similarly. For $G, H \in \mathcal{J}_2^{\mathrm{d}}$, we let $G \bullet H$ be the union of G and an isomorphic copy H' of H such that $src_G(2) = src_{H'}(1)$ and G and H' have nothing else in common. We let $src_{G\bullet H}(1) := src_G(1)$ and $src_{G\bullet H}(2) := src_{H'}(2)$. These operations are illustrated in Figure 1.2.

[4]The letter "J" in the notations $\mathcal{J}_2^{\mathrm{d}}$ and, below, in $\mathbb{J}_2^{\mathrm{d}}$, $\mathbb{JS}$ and the related notions refers to graphs that can have multiple edges. By contrast, the letter "G" used in the notations $\mathcal{G}^{\mathrm{u}}$ and, below, $\mathbb{G}^{\mathrm{u}}$, $\mathbb{GP}$ etc. refers to simple graphs. The subscript "2" refers to the two sources, and the superscript "d" to directed graphs.

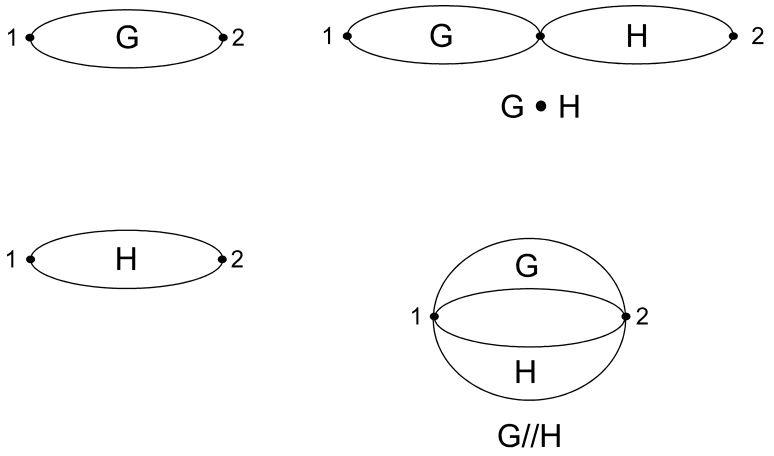

Figure 1.2: Series- and parallel-compositions.

The set of *series-parallel graphs*[5] is defined by the equation:

$$S \quad = \quad (S /\!\!/ S) \cup (S \bullet S) \cup \{e\} \tag{1.3}$$

where by "defined" we mean that S is the least subset of $\mathcal{J}_2^{\mathrm{d}}$ satisfying (1.3). As for cographs, this definition gives a notation of series-parallel graphs by terms. The set of terms is here $T(\{/\!\!/, \bullet, e\})$. Examples of terms are:

$$e,\ e /\!\!/ e,\ (e /\!\!/ e) \bullet (e /\!\!/ e),\ ((e /\!\!/ e) \bullet e) /\!\!/ (e \bullet e).$$

The graph denoted by the last of these terms is shown in Figure 1.3. Note that the subterm $(e /\!\!/ e) \bullet e$ denotes the three edges at the left of Figure 1.3, with their three incident vertices, whereas the subterm $e \bullet e$ denotes the two edges at the right, with their three incident vertices.

1.1.4 The general setting

Let F be a (functional) *signature*, that is, a set of *function symbols* such that each symbol f is given with a nonnegative integer intended to be the number of arguments of the corresponding function. This number is called its *arity* and is denoted by $\rho(f)$. A function symbol of arity 0 is also called a *constant symbol*.

An *F-algebra* $\mathbb{M}$ is a set M equipped with total functions $f_{\mathbb{M}} : M^{\rho(f)} \to M$ for all f in F. We write it $\mathbb{M} = \langle M, (f_{\mathbb{M}})_{f \in F} \rangle$. We call M the *domain* and $f_{\mathbb{M}}$ an *operation* of $\mathbb{M}$; if f has arity 0, then $f_{\mathbb{M}}$ is also called a *constant* of $\mathbb{M}$. The F-algebra $\mathbb{M}$ is *finite* if M is finite.

Let $X = \{x_1, \ldots, x_n\}$ be a set of *unknowns* (or variables), intended to denote subsets of M. A *polynomial* is an expression of the form $p = m_1 \cup \cdots \cup m_k$

[5]The term "series-parallel" is also used for partial orders ([*Möhr]) and, in a wider sense for undirected graphs without K_4 as a minor ([*Die]). Our series-parallel graphs are called *two-terminal series-parallel digraphs* in [*Möhr].

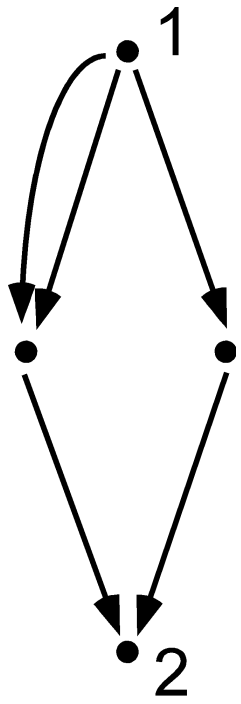

Figure 1.3: A series-parallel graph.

where each m_i is a *monomial*, i.e., a term written with the symbols of $F \cup X$ and well formed with respect to arities (the unknowns are of arity 0).

For each n-tuple $(L_1, \ldots, L_n)$ of subsets of M and each monomial m, the set $m(L_1, \ldots, L_n)$ is a subset of M. This subset is defined by taking $x_i = L_i$ and by interpreting each function symbol f as $f_{\mathbb{M}}$, where, for all $A_1, \ldots, A_{\rho(f)} \subseteq M$:

$$f_{\mathbb{M}}(A_1, \ldots, A_{\rho(f)}) := \{f_{\mathbb{M}}(a_1, \ldots, a_{\rho(f)}) \mid a_i \in A_i\}.$$

Hence $f_{\mathbb{M}}$ also denotes the extension to sets of the function $f_{\mathbb{M}} : M^{\rho(f)} \to M$. For a polynomial $p = m_1 \cup \cdots \cup m_k$ we define:

$$p(L_1, \ldots, L_n) := m_1(L_1, \ldots, L_n) \cup \cdots \cup m_k(L_1, \ldots, L_n).$$

A *system of polynomial equations* (an *equation system* for short) is a system of the form:

$$S = \langle x_1 = p_1, \ldots, x_n = p_n \rangle \tag{1.4}$$

where $p_1, \ldots, p_n$ are polynomials.

Example 1.2 In the particular case of the grammar G considered in Example 1.1, we let $F = \{\cdot, \varepsilon, a, b, c\}$ and $\mathbb{M} = \langle A^*, \cdot, \varepsilon, a, b, c \rangle$, where $A = \{a, b, c\}$ and $\cdot$ denotes concatenation; the equation system Σ_G can be written formally as follows:

$$\begin{cases} x_1 &= a \cdot (x_1 \cdot x_2) \cup x_1 \cdot x_1 \cup a \\ x_2 &= b \cdot ((x_2 \cdot x_1) \cdot x_2) \cup a \cup c \cup \varepsilon, \end{cases}$$

where the associativity of concatenation is not taken for granted any more. Note that for the constant symbol a of F we have $a_{\mathbb{M}} = a$ and also, according to the above extension, $a_{\mathbb{M}} = \{a\}$; similarly, the constant symbol ε denotes both the empty word ε and the language $\{\varepsilon\}$. □

Going back to the general case, a *solution* of a system S as in (1.4) is an n-tuple $(L_1, \ldots, L_n)$ in $\mathcal{P}(M)^n$ that satisfies the equations of S, which means that $L_i = p_i(L_1, \ldots, L_n)$ for every $i = 1, \ldots, n$. Solutions are compared by component-wise inclusion and every system has a least solution. The components of the least solutions of such systems are called the *equational sets* of the F-algebra $\mathbb{M}$. We will denote by $\mathbf{Equat}(\mathbb{M})$ the family of equational sets of $\mathbb{M}$.

For a signature F, we denote by $T(F)$ the set of terms written with the symbols of F and well formed with respect to arities. The usual notation for terms is with the function symbols in leftmost position, their arguments are between parentheses and separated by commas. In this notation, the term denoting the graph of Figure 1.3 is written $/\!\!/(\bullet(/\!\!/(e, e), e), \bullet(e, e))$.[6] As is well known, terms can be represented by certain labelled, directed and rooted trees.

[6] For associative binary operations the more readable infix notation will be used, although it is ambiguous as already observed. The infix notation of this term is $((e \mathbin{/\!\!/} e) \bullet e) \mathbin{/\!\!/} (e \bullet e)$.

This representation is the reason that terms are usually called trees in Formal Language Theory.

The set $T(F)$ is turned into an F-algebra, denoted by $\mathbb{T}(F)$, by defining the operation $f_{\mathbb{T}(F)}$ by

$$f_{\mathbb{T}(F)}(t_1, \ldots, t_{\rho(f)}) := f(t_1, \ldots, t_{\rho(f)}).$$

This operation performs no computation; it combines its arguments which are terms into a larger term.

For every F-algebra $\mathbb{M}$, a term $t \in T(F)$ has a *value* $t_{\mathbb{M}}$ in M that is formally defined as follows:

$$\begin{aligned} t_{\mathbb{M}} &:= f_{\mathbb{M}} \text{ if } t = f \text{ and } f \text{ has arity 0 (it is a constant symbol)}, \\ t_{\mathbb{M}} &:= f_{\mathbb{M}}(t_{1\mathbb{M}}, \ldots, t_{\rho(f)\mathbb{M}}) \text{ if } t = f(t_1, \ldots, t_{\rho(f)}). \end{aligned}$$

Since every term can be written in a unique way as f or $f(t_1, \ldots, t_{\rho(f)})$ for terms $t_1, \ldots, t_{\rho(f)}$, the value $t_{\mathbb{M}}$ of t is well defined. The mapping $t \mapsto t_{\mathbb{M}}$, also denoted by $val_{\mathbb{M}}$, is the unique F-algebra homomorphism from $\mathbb{T}(F)$ into $\mathbb{M}$.[7] An F-algebra $\mathbb{M}$ is *generated* by F if every element of M is the value of some term in $T(F)$.

An equation system S of the form (1.4) has a least solution in $\mathcal{P}(T(F))^n$ that is an n-tuple $(T_1, \ldots, T_n)$ of subsets of $T(F)$. The least solution $(L_1, \ldots, L_n)$ of S in $\mathcal{P}(M)^n$ is also characterized by $L_i = \{t_{\mathbb{M}} \mid t \in T_i\}$, for each $i = 1, \ldots, n$. This is an immediate consequence of a result of [MezWri] saying that the least fixed-point operator commutes with homomorphisms. A term t in T_i represents the structure of the element $t_{\mathbb{M}}$ of L_i as specified by the system S.

It follows in particular that for each i, $L_i = \emptyset$ if and only if $T_i = \emptyset$. Hence the least solutions of a system S in all algebras have the same empty components. The emptiness of each set T_i can be decided by the algorithm that decides the emptiness of a context-free language. Each set T_i is actually a context-free language over the alphabet consisting of F, parentheses and comma.

We will use these definitions for *algebras of graphs* $\mathbb{M}$ in the following way: M will be a class of graphs like $\mathcal{G}^{\mathrm{u}}$ or $\mathcal{J}_2^{\mathrm{d}}$ in the examples of cographs and series-parallel graphs, F will be a set of total functions $f_{\mathbb{M}} : M^{\rho(f)} \to M$ that will be used to construct graphs. These functions, called the *operations* of $\mathbb{M}$, generalize the concatenation of words. The *constants* will be basic graphs. For each such graph algebra $\mathbb{M}$, the class of equational sets $\mathbf{Equat}(\mathbb{M})$ generalizes the class of *context-free languages* since they are characterized as the components of the least solutions of equation systems as recalled in Section 1.1.1. There is thus no unique notion of a context-free set of graphs because this notion depends on the considered algebra.

However, even in the case of languages, several algebras can be considered, because one can enrich the monoid structure of A^* by new operations. This

[7] In general, a *homomorphism* from $\mathbb{N}$ to $\mathbb{M}$, where $\mathbb{N} = \langle N, (f_{\mathbb{N}})_{f \in F} \rangle$ is another F-algebra, is a mapping $h : N \to M$ such that for every $f \in F$ and all $n_1, \ldots, n_{\rho(f)} \in N$, we have $h(f_{\mathbb{N}}(n_1, \ldots, n_{\rho(f)})) = f_{\mathbb{M}}(h(n_1), \ldots, h(n_{\rho(f)}))$.

increases the class of equational sets, hence defines richer notions of context-free languages, if we take this term in the algebraic sense. The *squaring function* that associates with a word u the word uu, can be such an operation. Another one is the *shift* that associates with a word au the word ua, where a is a letter. The corresponding classes of equational sets have not received specific attention.

In the case of graphs, we will show that there are only two robust classes of equational sets, where *robust* means that they are closed under certain graph transformations definable by formulas of *monadic second-order logic*. These transformations, called *monadic second-order transductions*, play the role of rational transductions in the theory of formal languages.

Each of the two classes of context-free sets of graphs is the class of images of the set of finite binary trees under monadic second-order transductions of appropriate forms. Somewhat similarly, the class of context-free languages is the class of images of the language defined by the equation $L = aLbLc \cup \{d\}$, under all rational transductions. This language encodes binary trees. Hence trees play a major role in all three cases.

1.1.5 Derivation trees

Context-free grammars specify languages. However the real importance of the notion of a context-free grammar is that, when a word is recognized as well-formed, the grammar specifies one or several *parse trees* for this word. These trees are obtained as results of the *syntactic analysis* (or *parsing*) of the considered word. They represent the syntactical structures of the considered word as generated by the grammar. In compiling applications, grammars are constructed so as to be unambiguous, and each recognized word has a unique parse tree. This tree is the support of further computation, in particular type checking and translation into intermediate code.

Similarly, an equation system specifies a set of objects and, as we have seen, it additionally specifies terms that denote those objects and represent their structure. Let S be an equation system and $\mathbb{M} = \langle M, (f_{\mathbb{M}})_{f \in F} \rangle$ an algebra, and consider an algorithm that, for each element m of M, computes a term t that denotes m as specified by S (if such a term exists). Due to the similarity with context-free grammars, we will say that this is a *parsing algorithm* for S.

However, if we view a context-free grammar G such as the one of Example 1.1 as an equation system $S = \Sigma_G$ over the signature $F = \{\cdot, \varepsilon, a, b, c\}$, as indicated in Example 1.2, then the terms in $T(F)$ specified by the system Σ_G are not the parse trees of G (because they do not show which rules of G are applied). Nevertheless, it is possible to view G as an equation system S' in a different way, such that the terms of S' *do* correspond to the parse trees of G, or rather a variant of parse trees called *derivation trees*. Let us illustrate this for the context-free grammar G of Example 1.1.

Example 1.3 We consider again the grammar G of Example 1.1. Its rules are named by symbols $p, q, \ldots, w$, that we will consider as function symbols with arities defined by ρ such that $\rho(s) = 3$, $\rho(p) = \rho(q) = 2$, $\rho(r) = \rho(u) =$

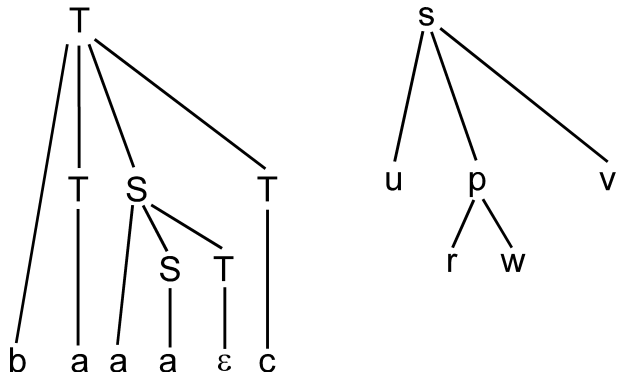

Figure 1.4: The parse tree of D and the term d.

$\rho(v) = \rho(w) = 0$; they form a signature P. The arity of a rule is the number of occurrences of nonterminals in the righthand side of the rule.

Consider for example the word *baaac* generated from nonterminal T by the derivation sequence D:

$$T \Rightarrow bTST \Rightarrow bTaSTT \Rightarrow baaSTT \Rightarrow baaaTT \Rightarrow baaaT \Rightarrow baaac$$

where the rules s, p, u, r, w, v are successively applied. (The arrow $\Rightarrow$ denotes the one-step derivation relation of G). Assuming that rule w is applied to the leftmost T in $baaaTT$, we associate with D the term $d = s(u, p(r, w), v)$ of $T(P)$. This term contains more information than the sequence (s, p, u, r, w, v); from it one can find all derivation sequences of the word *baaac* that are equivalent to D by permutations of steps. In particular the leftmost derivation sequence uses successively rules s, u, p, r, w, v, and the rightmost one uses rules s, v, p, w, r, u. Figure 1.4 shows the parse tree of D and the corresponding term d.

Terms like d will be called *derivation trees*. We keep the name *parse tree* for the trees like the one of Figure 1.4 (left part) that are used in the theory of parsing. (Good text books exposing this theory are the "Dragon Book" by Aho *et al.* [*AhoLSU] and the book by Crespi-Reghizzi [*Cre]).

The equation system Σ_G of Example 1.1 can be rewritten into the following system:

$$\Sigma'_G \begin{cases} K &= p(K, L) \cup q(K, K) \cup r \\ L &= s(L, K, L) \cup u \cup v \cup w. \end{cases}$$

Instead of solving this system for the F-algebra $\mathbb{M} = \langle \{a, b, c\}^*, \cdot, \varepsilon, a, b, c \rangle$ (which is the algebra for Σ_G in Example 1.2), we solve it for the P-algebra $\mathbb{M}'$ with the same domain $\{a, b, c\}^*$ but with the following interpretation of the symbols of P. If we interpret the symbols p, q, s by the following operations on

$A^* = \{a, b, c\}^*$ (where x, y, z denote words in A^*):

$$\begin{aligned} p(x, y) &:= axy, \\ q(x, y) &:= xy, \\ s(x, y, z) &:= bxyz, \end{aligned}$$

and the constant symbols r, u, v, w by the following words:

$$\begin{aligned} r &:= a, \\ u &:= a, \\ v &:= c, \\ w &:= \varepsilon, \end{aligned}$$

then Σ'_G is just an alternative writing of Σ_G, and its least solution for the algebra $\mathbb{M}'$ is also $(L(G, S), L(G, T))$. But the system Σ'_G has also a least solution (K', L') in $\mathcal{P}(T(P)) \times \mathcal{P}(T(P))$, and the derivation tree d is an element of L'. More generally we define the sets of *derivation trees of* G respectively associated with S and T as the sets of terms K' and L'. For the above interpretation of the symbols of P, we can evaluate every term t of $T(P)$ into a word $t_{\mathbb{M}'}$ in A^*. In particular $d_{\mathbb{M}'} = baaac$. Clearly, $L(G, S) = \{t_{\mathbb{M}'} \mid t \in K'\}$ and $L(G, T) = \{t_{\mathbb{M}'} \mid t \in L'\}$. Thus, since a parsing algorithm for Σ'_G produces derivation trees of G, it corresponds to a classical parsing algorithm of the context-free grammar G.

The system Σ'_G and the derivation trees of G represent the *abstract syntax* of the grammar G whereas the P-algebra $\mathbb{M}'$ represents its *concrete syntax*. It should be clear that the construction of Σ'_G and $\mathbb{M}'$ can be realized for every context-free grammar G. It should be noted, however, that the signature P and the algebra $\mathbb{M}'$ both depend on G.

A term that is associated with the word *baaac* according to the equation system Σ_G in Example 1.2, is $b \cdot ((a \cdot (a \cdot (a \cdot \varepsilon))) \cdot c)$. That term can be obtained from derivation tree d by (re)interpreting the symbols p, q, s as the following operations on terms in $T(\{\cdot, \varepsilon, a, b, c\})$: $p(x, y) := a \cdot (x \cdot y)$, $q(x, y) := x \cdot y$, and $s(x, y, z) := b \cdot ((x \cdot y) \cdot z)$. Thus, a parsing algorithm for Σ'_G (producing derivation trees) can easily be transformed into one for Σ_G (producing terms). □

In fact, derivation trees can be defined for the elements of general equational sets. The transformation of Σ_G into Σ'_G can be generalized into the transformation of an arbitrary equation system $S = \langle x_1 = p_1, \ldots, x_n = p_n \rangle$ into a system $S' = \langle x_1 = p'_1, \ldots, x_n = p'_n \rangle$ such that each polynomial p'_i is a union of monomials of the form $r(x_{i_1}, \ldots, x_{i_{\rho(r)}})$ corresponding one-to-one to the monomials of p_i, where r belongs to a signature P associated with S. If $m \in T(F \cup \{x_1, \ldots, x_n\})$ is the monomial of p_i to which $r(x_{i_1}, \ldots, x_{i_{\rho(r)}})$ corresponds, then $x_{i_1}, \ldots, x_{i_{\rho(r)}}$ is the sequence of unknowns that occur in m. Furthermore, we impose that each r has a unique occurrence in S'. The least solution of S' in $\mathcal{P}(T(P))^n$ defines the n-tuple of sets of *derivation trees* of S.

Let F be the signature over which S is written, and $\mathbb{M}$ the F-algebra for which S is to be solved. The function $M^{\rho(r)} \to M$ that interprets a symbol r in P is the one defined[8] (in the usual sense) by the unique term t_r in $T(F \cup \{y_1, \ldots, y_{\rho(r)}\})$ such that (i) the variables $y_1, \ldots, y_{\rho(r)}$ occur in t_r in that order and no variable y_j occurs more than once, and (ii) the monomial of S to which the monomial $r(x_{i_1}, \ldots, x_{i_{\rho(r)}})$ of S' corresponds is obtained by substituting x_{i_j} for y_j in the term t_r, for every $j = 1, \ldots, \rho(r)$.

We obtain thus a P-algebra $\mathbb{M}'$ (with the same domain as $\mathbb{M}$) and the system S' interpreted in $\mathbb{M}'$ is the same as the system S interpreted in $\mathbb{M}$.

The value mapping $t \mapsto t_{\mathbb{M}'}$ maps each set of derivation trees D_i (the i-th component of the least solution of S' in $\mathcal{P}(T(P))^n$) to the component L_i of the least solution of S in $\mathcal{P}(M)^n$. Taking $\mathbb{M} = \mathbb{T}(F)$, D_i is mapped to the set of terms T_i: the i-th component of the least solution of S in $\mathcal{P}(T(F))^n$. Thus, a parsing algorithm for S' can easily be transformed into one for S. Since a parsing algorithm for S' produces derivation trees that represent the syntactical structure of the elements of M as specified by the system S, it will also be called a parsing algorithm for S; thus, from now on, parsing algorithms produce derivation trees and/or terms.

Here is an example of the construction of S' from S.

Example 1.4 We let S be the system:

$$\begin{cases} x_1 &= x_2 \cup a \cup f(x_1, x_2, x_1) \\ x_2 &= h(g(x_1, x_1), a) \cup f(x_1, x_2, x_1). \end{cases}$$

Then S' is:

$$\begin{cases} x_1 &= r_{1,1}(x_2) \cup r_{1,2} \cup r_{1,3}(x_1, x_2, x_1) \\ x_2 &= r_{2,1}(x_1, x_1) \cup r_{2,2}(x_1, x_2, x_1) \end{cases}$$

and the functions that interpret $r_{i,j}$ are defined by the following terms:

$$\begin{aligned} t_{r_{1,1}} &:= y_1, \\ t_{r_{1,2}} &:= a, \\ t_{r_{1,3}} &:= f(y_1, y_2, y_3), \\ t_{r_{2,1}} &:= h(g(y_1, y_2), a), \\ t_{r_{2,2}} &:= f(y_1, y_2, y_3). \end{aligned}$$

Note that $r_{1,1}$ is interpreted as the identity function and that $r_{1,3}$ and $r_{2,2}$ are interpreted as the same function. The construction of S' from S does not depend on any F-algebra for which S is to be solved, and hence neither do the derivation trees of S.

[8] A function $M^k \to M$ defined by a term in $T(F \cup \{y_1, \ldots, y_k\})$ is a k-ary *derived operation* of $\mathbb{M}$.

1.2 Inductive sets of properties and recognizability

This section is a mild introduction to the algebraic notion of recognizability. This notion can be defined in several equivalent ways and we begin with its most concrete characterization, based on finite sets of properties that can be checked inductively.

1.2.1 Properties of the words of a context-free language

Let us consider the problem of proving an assertion of the form $\forall w \in L.\, P(w)$ where L is a context-free language, an equational set of graphs, or more generally, an equational set in an F-algebra with domain M, and where P is a property of the elements of M. Such an assertion expresses that P is universally valid on L.

Example 1.5 Let $X := \{f, x, y\}$ and $L \subseteq X^*$ be the language defined as the least solution (it is actually the unique solution) of the equation $L = fLL \cup \{x, y\}$. This language is the set of *Polish prefix notations* of the terms in $T(\{f, x, y\})$ where $\rho(f) = 2$ and $\rho(x) = \rho(y) = 0$. It satisfies the assertion $\forall w \in L.\, P(w)$ where property $P(w)$ is defined by:

$$2|w|_f = |w| - 1 \ \wedge \ \forall u \in X^*(u < w \Rightarrow 2|u|_f \geq |u|).$$

Here $|w|$ denotes the length of a word w, $|w|_f$ the number of occurrences of f in w and $<$ the strict prefix order on words. Establishing the following facts is a routine exercise:

(i) $P(w)$ holds if $w = x$ or $w = y$;

(ii) $P(w)$ holds if $w = fw_1w_2$ and $P(w_1)$ and $P(w_2)$ both hold.

Then the proof that $\forall w \in L.\, P(w)$ holds can be done by induction on the length of a *derivation sequence* of a word w in L, relative to the context-free grammar with rules $A \to fAA$, $A \to x$ and $A \to y$.

However this proof can also be formulated in terms of equation systems. By the Least Fixed-Point Theorem (Theorem 3.7), the least solution in $\mathcal{P}(M)^n$ of a system S of the form (1.4) is also the least solution of the corresponding system of inclusions:

$$\begin{cases} L_1 \supseteq p_1(L_1, \ldots, L_n) \\ \vdots \qquad\qquad \vdots \\ L_n \supseteq p_n(L_1, \ldots, L_n) \end{cases} \tag{1.5}$$

where $L_i \subseteq M$. In the above example, Facts (i) and (ii) can be restated as the inclusion:

$$K_P \supseteq fK_PK_P \cup \{x, y\} \tag{1.6}$$

where $K_P := \{w \in X^* \mid P(w)\}$. Since the language L, defined by $L = fLL \cup \{x, y\}$, is also the least solution of (1.6), we have $L \subseteq K_P$ which yields the validity of $\forall w \in L.\, P(w)$.

We will say that an assertion of the form $\forall w \in L.\, P(w)$ is *provable by fixed-point induction* in order to express that this method applies, i.e., that property P satisfies Facts (i) and (ii). A property Q may be universally valid on the language L without this being provable by fixed-point induction. For example consider the property $Q(w)$ defined for $w \in X^*$ by $|w| = 1$ or $w \neq \widetilde{w}$ (where $\widetilde{w}$ denotes the mirror image of w) and $K_Q := \{w \in X^* \mid Q(w)\}$. It is not true that $fK_QK_Q \subseteq K_Q$ (since x and f belong to K_Q and $fxf \notin K_Q$). However $L \subseteq K_Q$. Hence the valid assertion $\forall w \in L.\, Q(w)$ is not provable by fixed-point induction.

In order to establish that Q is universally valid on L it suffices to find a property R such that:

(1) $\forall w \in X^*(R(w) \Rightarrow Q(w))$ is true, and

(2) $\forall w \in L.\, R(w)$ is provable by fixed-point induction.

We can take $R(w) :\Longleftrightarrow w \in \{x, y\} \cup fX^*\{x, y\}$. We prove in this way a stronger assertion than $\forall w \in L.\, Q(w)$ which was the initial goal.

Finding such R is always possible in a trivial way, by taking $R(w)$ to mean that w belongs to L, which does not yield any proof since (1) is just what is to be proved and (2) holds in a trivial way. Hence this observation is interesting if R can be found such that (1) and (2) are "easily provable", which is not a rigorous notion. A proof method can be defined by requiring that R is expressed in a particular language and/or that the proofs of (1) and (2) can be done in a particular proof system. We will give below an example of such a situation (Proposition 1.6). □

We generalize the notion of an assertion provable by fixed-point induction to systems of equations. Let S be an equation system of the general form (1.4) and let $(L_1, \ldots, L_n)$ be its least solution in $\mathcal{P}(M)^n$ for some F-algebra $\mathbb{M}$. Let $(P_i)_{1 \leq i \leq n}$ be an n-tuple of properties of elements of M, such that the assertion

$$\forall i \in [n], \forall d \in L_i.\, P_i(d) \tag{1.7}$$

is true. We say that this assertion is *provable by fixed-point induction* if:

$$K_{P_i} \supseteq p_i(K_{P_1}, \ldots, K_{P_n}) \tag{1.8}$$

for each $i = 1, \ldots, n$, where K_{P_i} denotes $\{d \in M \mid P_i(d)\}$. It follows from the Least Fixed-Point Theorem that the validity of (1.8) for all i implies that $L_i \subseteq K_{P_i}$ for all i, hence that the considered assertion (1.7) is true. The proof method consisting in proving (1.8) to establish (1.7) is thus sound.

Let us go back to context-free languages. Let the context-free language $L_1 \subseteq A^*$ be defined as the first component of the least solution $(L_1, \ldots, L_n)$ of an equation system $S = \langle x_1 = p_1, \ldots, x_n = p_n \rangle$.

Proposition 1.6 For every regular language K such that $L_1 \subseteq K$, there exists an n-tuple of regular languages $(K_1, \ldots, K_n)$ such that $K_1 \subseteq K$ and such that the assertion:

$$\forall i \in [n], \forall d \in L_i.\, d \in K_i$$

is provable by fixed-point induction.

Proof sketch: We have $K = h^{-1}(N)$ where h is a homomorphism of A^* into a finite monoid[9] $\mathbb{Q} = \langle Q, \cdot_{\mathbb{Q}}, 1_{\mathbb{Q}} \rangle$ and $N \subseteq Q$ ([*Eil, *Sak]). We define $K_i := h^{-1}(h(L_i)) = \bigcup\{h^{-1}(q) \mid q \in Q,\ h^{-1}(q) \cap L_i \neq \emptyset\}$. Note that this definition does not depend on N.

We first prove that $K_1 \subseteq K$. We consider a word v in K_1. For some word v' in L_1, we have $h(v) = h(v')$. Since $L_1 \subseteq K$, we have $v' \in K$. Hence $v \in K$ since $h(v') \in N$ and $K = h^{-1}(N)$. This achieves the first goal.

In order to prove that for each i we have $p_i(K_1, \ldots, K_n) \subseteq K_i$, we need only consider a monomial α of p_i (i.e., the righthand side of a rule $x_i \to \alpha$ of the context-free grammar S) and prove that:

$$w_0 K_{i_1} w_1 \cdots K_{i_k} w_k \subseteq K_i, \tag{1.9}$$

where $\alpha = w_0 x_{i_1} w_1 x_{i_2} \cdots x_{i_k} w_k$ with $w_0, \ldots, w_k \in A^*$. Let $v_j \in K_{i_j}$ for $j = 1, \ldots, k$. There exist $v'_1, \ldots, v'_k$ such that $h(v'_j) = h(v_j)$ and $v'_j \in L_{i_j}$ for each j. Hence the word $v' = w_0 v'_1 w_1 \cdots v'_k w_k$ belongs to L_i, because $(L_1, \ldots, L_n)$ is a solution of S. Letting $v = w_0 v_1 w_1 \cdots v_k w_k$ we have $h(v) = h(v')$ hence $v \in K_i$. This proves inclusion (1.9) and completes the proof of the proposition. ■

This result shows that assertions of the form $L \subseteq K$ where L is a context-free language and K is a regular one can be proved by fixed-point induction. The proofs that a "guessed" n-tuple $(K'_1, \ldots, K'_n)$ satisfies $K'_1 \subseteq K$ and the inclusions $K'_i \supseteq p_i(K'_1, \ldots, K'_n)$ establish that $L_1 \subseteq K$ and use only algorithms on finite automata: Boolean operations, concatenation and emptiness test.

1.2.2 Some properties of series-parallel graphs

We now show that fixed-point induction can also be used for proving universal properties of equational sets of graphs. We use the example of the set of series-parallel graphs defined by Equation (1.3) considered in Section 1.1.3:

$$S \quad = \quad (S /\!\!/ S) \cup (S \bullet S) \cup \{e\}$$

[9] $\mathbb{Q} = \langle Q, \cdot_{\mathbb{Q}}, 1_{\mathbb{Q}} \rangle$ is a monoid if the binary operation $\cdot_{\mathbb{Q}}$ is associative with $1_{\mathbb{Q}}$ as unit element.

where $S \subseteq \mathcal{J}_2^{\mathrm{d}}$. We will prove the assertions $\forall G \in S.\, P_i(G)$, where the properties P_i are defined as follows:

$$\begin{aligned} P_1(G) &:\Longleftrightarrow G \text{ is connected,} \\ P_2(G) &:\Longleftrightarrow G \text{ is bipolar,} \\ P_3(G) &:\Longleftrightarrow G \text{ is planar,} \\ P_4(G) &:\Longleftrightarrow G \text{ has no directed cycle.} \end{aligned}$$

A directed graph G with two sources denoted by $src_G(1)$ and $src_G(2)$ (cf. Section 1.1.3) is *bipolar* if it has no directed cycle and every vertex belongs to a directed path from $src_G(1)$ to $src_G(2)$.

Following the same method as for the language L of Example 1.5, we need only prove that $P_i(e)$ holds and that, for all graphs G, H in $\mathcal{J}_2^{\mathrm{d}}$: $P_i(G) \wedge P_i(H)$ implies $P_i(G /\!\!/ H) \wedge P_i(G \bullet H)$.

These facts are clearly true for properties P_1 and P_2. The assertions that every graph in S satisfies P_1 on the one hand and P_2 on the other are thus provable by fixed-point induction, hence P_1 and P_2 are universally valid on S.

Property P_3 is not provable in this way because it is not true that, for all graphs G, H in $\mathcal{J}_2^{\mathrm{d}}$, $P_3(G) \wedge P_3(H)$ implies $P_3(G /\!\!/ H)$. For a counterexample, take H to be an edge, G to be K_5 minus one edge (K_5 is a complete simple undirected graph with 5 vertices) and equipped with sources in such a way that $G /\!\!/ H$ is isomorphic to K_5, which is non-planar. However, G and H are both planar, hence satisfy P_3.

For proving that every series-parallel graph is planar, we can use the property Q saying that a graph has a planar drawing with its two sources on the external face. (The books [*Die] and [*MohaTho] give formal definitions about graphs on surfaces.) This property is provable by fixed-point induction (with respect to the equation defining S), hence it is true for all graphs in S. Since $Q(G)$ implies $P_3(G)$ for all graphs G in $\mathcal{J}_2^{\mathrm{d}}$, we obtain the announced result.

The case of property P_4 (the absence of directed cycles) is similar. The assertion that every graph in S satisfies P_4 is not provable by fixed-point induction, however it is true. For proving it, one takes the stronger property P_2 considered above.

This proof technique can be applied to systems of equations (and not only to single equations) and to graph properties expressed in monadic second-order logic (see Section 1.3). More precisely, for every such graph property P and every equational set of graphs L:

(1) we can decide whether or not P is universally valid on L,

(2) if P is, then we can build a set of auxiliary properties, like the sets $K_1, \ldots, K_n$ in Proposition 1.6, that yields a proof by fixed-point induction of the universal validity of P on L.

These constructions and the verification of conditions like (1.8) can be done by algorithms.

1.2.3 Inductive sets of properties

We now consider properties that are not necessarily universally valid on the considered sets of graphs, so that they raise decision problems. We will consider a graph property as a function from graphs to $\{True, False\}$.

Example 1.7 Cographs.

The property $E(G)$ that a cograph G (cographs are defined in Section 1.1.2) has an even number of vertices is not universally valid. However, it satisfies the following rules:

$$\begin{aligned} E(\mathbf{1}) &= True, \\ E(G \oplus H) &= (E(G) \Leftrightarrow E(H)), \\ E(G \otimes H) &= (E(G) \Leftrightarrow E(H)). \end{aligned}$$

For Boolean values p and q, $p \Leftrightarrow q$ is defined as usual as $((p \wedge q) \vee (\neg p \wedge \neg q))$. It follows that if a cograph G is the value of a term t in $T(\{\oplus, \otimes, \mathbf{1}\})$ (we denote this by $G = val(t)$), then the validity of $E(G)$ can be determined by computing $E(val(t'))$ for all subterms t' of t, by starting from the smallest ones. This type of computation can be done by automata on terms that we will present in Section 1.2.4.

The property $F(G)$ defined as "G has an even number of edges", can be checked in a similar way by computing simultaneously $E(val(t'))$ and $F(val(t'))$ for every subterm t' of t. This computation uses the following rules:

$$\begin{aligned} F(\mathbf{1}) &= True, \\ F(G \oplus H) &= (F(G) \Leftrightarrow F(H)), \\ F(G \otimes H) &= \Big(\Big(F(G) \Leftrightarrow F(H)\Big) \wedge \Big(E(G) \vee E(H)\Big)\Big) \vee \\ &\quad \Big(\Big(F(G) \Leftrightarrow \neg F(H)\Big) \wedge \Big(\neg E(G) \wedge \neg E(H)\Big)\Big). \end{aligned}$$

Hence, $F(G)$ can be checked with the help of $E(G)$ as additional information. □

We now generalize this computation method. We introduce a definition relative to an arbitrary F-algebra $\mathbb{M}$. Let $\mathcal{P}$ be a set of properties, i.e., of mappings: $M \to \{True, False\}$. We say that $\mathcal{P}$ is *F-inductive* if for every $P \in \mathcal{P}$, for every $f \in F$ of arity $n > 0$, and for every $m_1, \ldots, m_n$ in M, the Boolean value $P(f_{\mathbb{M}}(m_1, \ldots, m_n))$ can be computed by a fixed Boolean expression depending on P and f, in terms of finitely many Boolean values $Q(m_i)$ with Q in $\mathcal{P}$ and $i = 1, \ldots, n$.

In the previous example, the set of properties $\{E, F\}$ is $\{\oplus, \otimes\}$-inductive for the algebra of cographs, but the set $\{F\}$ is not. The computation of $F(G \otimes H)$ can be expressed by:

$$F(G \otimes H) = B(E(G), F(G), E(H), F(H))$$

where $B(p_1, p_2, p_3, p_4)$ is the Boolean expression:

$$\Big((p_2 \Leftrightarrow p_4) \wedge (p_1 \vee p_3)\Big) \vee \Big((p_2 \Leftrightarrow \neg p_4) \wedge (\neg p_1 \wedge \neg p_3)\Big).$$

In order to have a uniform notation, if $\mathcal{P}$ is finite and is enumerated as $\{P_1, \ldots, P_k\}$, we write:

$$P_i(f_{\mathbb{M}}(m_1, \ldots, m_n)) =$$
$$B_{i,f}\Big(P_1(m_1), \ldots, P_k(m_1), P_1(m_2), \ldots, P_k(m_2), \ldots, P_1(m_n), \ldots, P_k(m_n)\Big)$$

to formalize the way $P_i(f_{\mathbb{M}}(m_1, \ldots, m_n))$ can be computed from the Boolean values $P_i(m_j)$. In this writing, each $B_{i,f}$ is a Boolean expression in the propositional variables $p_{h,j}$, $1 \leq h \leq k$, $1 \leq j \leq n$, and $P_h(m_j)$ is substituted in $B_{i,f}$ for $p_{h,j}$.

An important theorem is the following one:

Theorem 1.8 (Filtering Theorem, concrete version)
Let $\mathbb{M}$ be an F-algebra and $\mathcal{P}$ be a finite F-inductive set of properties. For every equational set L of $\mathbb{M}$, for every P in $\mathcal{P}$, the set $L_P := \{x \in L \mid P(x)\}$ is equational. If the Boolean expressions involved in the definition of the inductivity of $\mathcal{P}$ are given, then the construction of a system of equations defining L_P from one defining L is effective, i.e., can be done by an algorithm. □

The classical result saying that the intersection of a context-free language and a regular one is context-free is a special case of this theorem. Let us consider the language L of Example 1.5 defined by the equation $L = fLL \cup \{x, y\}$. From this language we want to keep only the words whose length is a multiple of 3. For $i \in \{0, 1, 2\}$, we let $L_i := \{w \in L \mid \mathrm{mod}_3(|w|) = i\}$. The triple (L_0, L_1, L_2) is the least solution (and actually also the unique one) of the system:

$$\begin{cases} L_0 &= fL_0L_2 \cup fL_1L_1 \cup fL_2L_0, \\ L_1 &= fL_0L_0 \cup fL_1L_2 \cup fL_2L_1 \cup \{x, y\}, \\ L_2 &= fL_0L_1 \cup fL_1L_0 \cup fL_2L_2. \end{cases}$$

It follows that the language L_0 is context-free. A similar example for cographs is System (1.2) in Section 1.1.2 (cf. the discussion after (1.2)).

Corollary 1.9 Let $\mathbb{M}$ and $\mathcal{P}$ be as in Theorem 1.8. For every equational set L of $\mathbb{M}$, one can decide whether or not a property P in $\mathcal{P}$ is universally valid on L, and whether or not it is satisfied by some element of L.

Proof sketch: We assume that L is given by a system of equations S. By using Theorem 1.8 we can construct a system S' that defines L_P. As noted in Section 1.1.4, we can test the emptiness of the components of the least solution of S', hence in particular of L_P. We can thus decide if P is satisfied by some element of L.

Since $\mathcal{P}$ is inductive, so is $\mathcal{P} \cup \{\neg P\}$. We can apply the previous result to $\neg P$. Then P is universally valid on L if and only if $L_{\neg P} = \emptyset$, which is decidable. ■

Example 1.10 We again let L be the language of Example 1.5 defined by the equation $L = fLL \cup \{x, y\}$. We know from this example that every word of L has odd length (because $|w| = 2|w|_f + 1$ for every $w \in L$), but we will see how the algorithm of Corollary 1.9 "discovers" this fact. Let K_0 and K_1 be the sets of words in L of even and odd length respectively. These languages are defined by the two equations:

$$\begin{cases} K_0 &= fK_0K_1 \cup fK_1K_0 \\ K_1 &= fK_0K_0 \cup fK_1K_1 \cup \{x, y\}. \end{cases}$$

It is easy to see that K_0 is empty (just look at the corresponding context-free grammar). Hence $K_1 = L$ and every word of L has odd length. □

It is useful to have a proof by fixed-point induction that a property is universally valid on an equational set although an algorithm can also give the answer, because a proof is more informative than the yes or no answer of an algorithm: it shows the properties of all components of the solution of the equation system that "contribute" to the validity of the proved property. This is clear in the case of Proposition 1.6.

We now consider one more example about graphs.

Example 1.11 The 2-colorability of series-parallel graphs.

We illustrate Theorem 1.8 with the 2-colorability of series-parallel graphs. A *proper vertex k-coloring* of a graph assigns to each vertex a *color*, i.e., an element of $\{1, \ldots, k\}$ such that two adjacent vertices have different colors. A graph is *k-colorable* if it has a proper vertex k-coloring. We consider three properties of a series-parallel graph G defined as follows:

$$\begin{array}{lcl} \gamma_2(G) & :\Longleftrightarrow & G \text{ is 2-colorable,} \\ \sigma(G) & :\Longleftrightarrow & G \text{ is 2-colorable with the two sources of the same color,} \\ \delta(G) & :\Longleftrightarrow & G \text{ is 2-colorable with the two sources of different colors.} \end{array}$$

The set of series-parallel graphs is defined by Equation (1.3)

$$S \quad = \quad (S /\!\!/ S) \cup (S \bullet S) \cup \{e\}$$

in the algebra $\langle \mathcal{J}_2^{\mathrm{d}}, /\!\!/, \bullet, e \rangle$.

Property γ_2 is not universally valid on S because $\gamma_2(e /\!\!/ (e \bullet e)) = \mathit{False}$.[10] The set $\{\gamma_2\}$ is not inductive because $\gamma_2(e) = \mathit{True}$, $\gamma_2(e \bullet e) = \mathit{True}$, $\gamma_2(e /\!\!/ e) = \mathit{True}$ and $\gamma_2(e /\!\!/ (e \bullet e)) = \mathit{False}$. It follows that the validity of $\gamma_2(G /\!\!/ H)$ cannot be deduced from those of $\gamma_2(G)$ and $\gamma_2(H)$. Hence, as in the case of property F for cographs in Example 1.7, we need additional properties. They will be σ and

[10] One can prove by fixed-point induction that every series-parallel graph is 3-colorable in such a way that its two sources have different colors.

δ. The set $\{\sigma, \delta\}$ is inductive; this is clear from the following facts:

$$\begin{array}{lcl} \sigma(G \bullet H) & = & (\sigma(G) \wedge \sigma(H)) \vee (\delta(G) \wedge \delta(H)), \\ \delta(G \bullet H) & = & (\delta(G) \wedge \sigma(H)) \vee (\sigma(G) \wedge \delta(H)), \\ \sigma(G /\!\!/ H) & = & \sigma(G) \wedge \sigma(H), \\ \delta(G /\!\!/ H) & = & \delta(G) \wedge \delta(H). \end{array} \tag{1.10}$$

One can thus compute for every term t in $T(\{/\!\!/, \bullet, e\})$ the pair of Boolean values $(\sigma(val(t)), \delta(val(t)))$ by induction on the structure of t (where $val(t)$ is the graph in $\mathcal{J}_2^{\mathrm{d}}$ defined by a term t). From the pair of Boolean values associated with a term t such that $val(t) = G$, one can decide whether $\gamma_2(G)$ holds or not, since for every graph G in $\mathcal{J}_2^{\mathrm{d}}$, $\gamma_2(G)$ is equivalent to $\sigma(G) \vee \delta(G)$. This computation can be formalized as the run of a finite deterministic automaton on t because the considered inductive set of properties, here $\{\sigma, \delta\}$, is finite. The finiteness of a set of inductive properties is the essence of the notion of recognizability that we now introduce informally.

1.2.4 Recognizability

In Formal Language Theory, the *recognizability* (or *regularity*) of a set of finite or infinite words or terms means frequently that this set is defined by a finite automaton of some kind. Recognizable sets of finite words and terms are defined by finite *deterministic* automata and this fact yields algebraic characterizations of recognizability in terms of homomorphisms into finite algebras. In particular, a language $L \subseteq A^*$ is recognizable if and only if $L = h^{-1}(N)$ where $h : A^* \to \mathbb{Q}$ is a monoid homomorphism, $\mathbb{Q}$ is a finite monoid and $N \subseteq Q$. (We have used this fact in the proof of Proposition 1.6).

This characterization has the advantage of being extendable in a meaningful way to any algebra, whereas the notion of automaton has no immediate generalization to arbitrary algebras. Furthermore, it fits very well with the notion of an equational set. The Filtering Theorem shows this, as we will explain in Section 1.2.5.

Following Mezei and Wright [MezWri], we say that a subset L of an F-algebra $\mathbb{M}$ (where F is finite) is *recognizable* if $L = h^{-1}(N)$ for some homomorphism of F-algebras $h : \mathbb{M} \to \mathbb{Q}$ where $\mathbb{Q}$ is a finite F-algebra and $N \subseteq Q$. We will denote by $\mathbf{Rec}(\mathbb{M})$ the family of recognizable subsets of $\mathbb{M}$.

The above definition of recognizability of L is equivalent to saying that the property P^L of the elements of M such that $P^L(x)$ is *True* if and only if $x \in L$, belongs to a finite F-inductive set of properties. In fact, assume that L is recognizable and let $Q = \{q_1, \ldots, q_k\}$. Define for each $i \in [k]$ the property P_i by: $P_i(m) :\Longleftrightarrow h(m) = q_i$. Since $h(f_{\mathbb{M}}(m_1, \ldots, m_n)) = f_{\mathbb{Q}}(h(m_1), \ldots, h(m_n))$, the Boolean value $P_i(f_{\mathbb{M}}(m_1, \ldots, m_n))$ can be computed from the Boolean values $P_h(m_j)$, by a Boolean expression. Thus, $\{P_1, \ldots, P_k\}$ is F-inductive in $\mathbb{M}$. Hence, also the set $\{P^L, P_1, \ldots, P_k\}$ is inductive, because $P^L(m) = \bigvee_{q_i \in N} P_i(m)$. The other direction of the equivalence is discussed in Section 1.2.5.

Proposition 1.6 also holds in general for recognizable sets instead of regular languages (with exactly the same proof, replacing A^* by $\mathbb{M}$). Thus, inclusions $L \subseteq K$ with $L \in \mathbf{Equat}(\mathbb{M})$ and $K \in \mathbf{Rec}(\mathbb{M})$ are provable by fixed-point induction, using auxiliary properties that correspond to recognizable subsets of M. Together with Corollary 1.9, this shows that the analogues of statements (1) and (2) at the end of Section 1.2.2 hold for properties that correspond to recognizable sets, in arbitrary algebras.

We now recall the link between recognizability for subsets of $T(F)$ and finite automata on terms (i.e., bottom-up finite tree automata). Let us consider a set $L \in \mathbf{Rec}(\mathbb{T}(F))$ defined as $L = h^{-1}(N)$ where h is the unique homomorphism: $\mathbb{T}(F) \to \mathbb{Q}$, $\mathbb{Q}$ is a finite F-algebra and $N \subseteq Q$. (Note that $h(t) = t_{\mathbb{Q}}$). The pair $(\mathbb{Q}, N)$ corresponds to a finite deterministic and complete *F-automaton* $\mathscr{A}(\mathbb{Q}, N)$ (the general definitions can be found in the book [*Com+] and in the book chapter [*GecSte], Section 3.3) with set of states Q, set of accepting states N and transitions consisting of the tuples $(a_1, \ldots, a_{\rho(f)}, f, a)$ such that $f \in F$, $a_1, \ldots, a_{\rho(f)}, a \in Q$ and $a = f_{\mathbb{Q}}(a_1, \ldots, a_{\rho(f)})$.

On each term t in $T(F)$, the automaton $\mathscr{A} = \mathscr{A}(\mathbb{Q}, N)$ has a unique ("bottom-up") *run*, defined as a mapping $run_{\mathscr{A},t} : Pos(t) \to Q$ such that $run_{\mathscr{A},t}(u) = h(t/u)$ for every position[11] u of t. Hence t is accepted by $\mathscr{A}$ if and only if $h(t) = run_{\mathscr{A},t}(root_t) \in N$.

Conversely, if L is the set of terms in $T(F)$ accepted by a finite, possibly not deterministic, automaton $\mathscr{B}$, then it is also accepted by a finite deterministic and complete automaton $\mathscr{A}$ (that one can construct from $\mathscr{B}$) and there is a unique pair $(\mathbb{Q}, N)$ such that $\mathscr{A}(\mathbb{Q}, N) = \mathscr{A}$. Hence, L is recognizable in $\mathbb{T}(F)$.

By a *recognizable set of graphs*, we will mean a subset of a graph algebra that is recognizable with respect to that algebra. No notion of "graph automaton" arises from this definition. However, we obtain finite automata accepting the sets of terms that denote the graphs of recognizable sets (this is true because the signature is finite). We will give a more precise statement in Theorem 1.12 below.

1.2.5 From inductive sets to automata

Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a finite inductive set of properties on an F-algebra $\mathbb{M}$ where F is finite. We associate with $\mathcal{P}$ a finite deterministic and complete F-automaton $\mathscr{A} = \mathscr{A}(\mathbb{Q}, N)$ as follows. Its set of states is $Q = \{True, False\}^k$; its transitions, i.e., the operations of $\mathbb{Q}$, are defined in such a way that for every f in F of arity n, we have: $f_{\mathbb{Q}}(q_1, \ldots, q_n) = q$ if and only if $q_i = (a_{1,i}, \ldots, a_{k,i})$, $q = (b_1, \ldots, b_k)$ belong to $\{True, False\}^k$ and (we use the notation introduced in Section 1.2.3):

$$b_i = B_{i,f}(a_{1,1}, \ldots, a_{k,1}, a_{1,2}, \ldots, a_{k,2}, \ldots, a_{1,n}, \ldots, a_{k,n}).$$

It follows that for every $t \in T(F), t_{\mathbb{Q}} = (P_1(t_{\mathbb{M}}), \ldots, P_k(t_{\mathbb{M}})) \in \{True, False\}^k$.

[11] The set $Pos(t)$ of positions of t is the set of occurrences of the symbols of F. We denote by t/u the subterm of t issued from u and by $root_t$ the first position of t. Formal definitions are in Chapter 2.

Hence if we want to specify by an automaton the set of objects $m \in M$ that satisfy $P_3(m)$ (to take an example), we take as set N of accepting states the set of Boolean vectors $(b_1, \ldots, b_k)$ such that $b_3 = \mathit{True}$. More precisely, a term $t \in T(F)$ is accepted by $\mathscr{A}$ if and only if $t_{\mathbb{M}}$ has property P_3.

This proves the implication (2) $\Rightarrow$ (3) in the next result. The other direction also holds, provided $\mathbb{M}$ is generated by F (as defined in Section 1.1.4).

Theorem 1.12 Let $\mathbb{M}$ be an F-algebra generated by F, where F is finite, and let $L \subseteq M$. The following are equivalent:

(1) L is recognizable in $\mathbb{M}$,

(2) $L = \{m \in M \mid P(m)\}$ where P belongs to a finite F-inductive set of properties,

(3) the set of terms t in $T(F)$ such that $t_{\mathbb{M}}$ belongs to L is recognizable in $\mathbb{T}(F)$, equivalently, is the set accepted by a finite F-automaton. □

The equivalence of (1) and (2) gives a concrete meaning to the algebraic notion of recognizability (and does not need $\mathbb{M}$ being generated by F). The implication (1) $\Rightarrow$ (2) was shown in Section 1.2.4. The other direction follows from the above discussion. If $h : M \to Q$ is defined by $h(m) := (P_1(m), \ldots, P_k(m))$, then h is a homomorphism from $\mathbb{M}$ to the finite F-algebra $\mathbb{Q}$, because $\mathcal{P}$ is inductive. And, e.g., $h^{-1}(N) = \{m \in M \mid P_3(m)\}$.

Let L be an equational set of $\mathbb{M}$, defined by an equation system $S = \langle x_1 = p_1, \ldots, x_n = p_n\rangle$, and assume that $L \subseteq \{m \in M \mid P_3(m)\}$. The equivalence of (1) and (2) implies that such an inclusion is provable by fixed-point induction, using auxiliary properties $R_1, \ldots, R_n$ that belong to a finite inductive set of properties. In fact, considering the proof of Proposition 1.6, with the above definition of $\mathbb{Q}$, N and h, it can be seen that every R_i is a Boolean combination of $P_1, \ldots, P_k$ and hence $\mathcal{P} \cup \{R_1, \ldots, R_n\}$ is an inductive set of properties.

The equivalence of (1) and (3) implies that the membership in a recognizable set of an element of M, given as $t_{\mathbb{M}}$, for some term t in $T(F)$, or the validity of $P(t_{\mathbb{M}})$ where P belongs to a finite inductive set of properties, can be checked in time $O(|t|)$, i.e., in time linear in the size of t.

Let us go back to Example 1.11 about the 2-colorability of series-parallel graphs. For the inductive set $\mathcal{P} = \{\sigma, \delta\}$ we obtain the set of states

$$Q = \{\mathit{True}, \mathit{False}\} \times \{\mathit{True}, \mathit{False}\} = \{(\sigma, \delta), (\sigma, \overline{\delta}), (\overline{\sigma}, \delta), (\overline{\sigma}, \overline{\delta})\},$$

where, for readability, we use σ and δ to denote *True* and $\overline{\sigma}$ and $\overline{\delta}$ to denote *False*. For every state $q \in Q$ we let S_q be the set of series-parallel graphs G such that $(\sigma(G), \delta(G)) = q$, i.e., $S_q = h^{-1}(q) \cap S$ with $h(G) = (\sigma(G), \delta(G))$ (cf. the proof of Proposition 1.6). Thus, $S_{\sigma,\delta}$ is the set of series-parallel graphs that satisfy σ and δ, $S_{\sigma,\overline{\delta}}$ the set of those that satisfy σ and not δ, $S_{\overline{\sigma},\delta}$ the set of those that satisfy δ and not σ, and $S_{\overline{\sigma},\overline{\delta}}$ the set of those that satisfy neither σ nor δ. From Properties (1.10) we obtain the operations $/\!\!/_{\mathbb{Q}}$, $\bullet_{\mathbb{Q}}$ and the constant $e_{\mathbb{Q}}$, which determine the transitions of the automaton $\mathscr{A}$. For example, since

the graph defined by e satisfies δ and not σ, we have $e_{\mathbb{Q}} = (\overline{\sigma}, \delta)$. As another example, if $G = H \bullet K$ and both H and K satisfy δ and not σ, then G satisfies σ and not δ; hence $\bullet_{\mathbb{Q}}((\overline{\sigma}, \delta), (\overline{\sigma}, \delta)) = (\sigma, \overline{\delta})$.

From the defining equation $S = (S /\!\!/ S) \cup (S \bullet S) \cup e$ and the transitions of $\mathscr{A}$ we obtain the following system of equations that define the sets $S_{\sigma,\delta}$, $S_{\overline{\sigma},\delta}$, $S_{\sigma,\overline{\delta}}$ and $S_{\overline{\sigma},\overline{\delta}}$ (we omit parentheses around terms like $S_{\sigma,\delta} /\!\!/ S_{\sigma,\delta}$ for better readability):

$$
\begin{aligned}
(a)\quad S_{\sigma,\delta} &= S_{\sigma,\delta} /\!\!/ S_{\sigma,\delta} \cup \\
&\quad S_{\sigma,\delta} \bullet S_{\sigma,\delta} \cup S_{\sigma,\delta} \bullet S_{\sigma,\overline{\delta}} \cup S_{\sigma,\delta} \bullet S_{\overline{\sigma},\delta} \cup \\
&\quad S_{\sigma,\overline{\delta}} \bullet S_{\sigma,\delta} \cup S_{\overline{\sigma},\delta} \bullet S_{\sigma,\delta} \\
(b)\quad S_{\overline{\sigma},\delta} &= e \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\sigma,\delta} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\sigma,\delta} \cup \\
&\quad S_{\sigma,\overline{\delta}} \bullet S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\delta} \bullet S_{\sigma,\overline{\delta}} \\
(c)\quad S_{\sigma,\overline{\delta}} &= S_{\sigma,\delta} /\!\!/ S_{\sigma,\overline{\delta}} \cup S_{\sigma,\overline{\delta}} /\!\!/ S_{\sigma,\delta} \cup S_{\sigma,\overline{\delta}} /\!\!/ S_{\sigma,\overline{\delta}} \cup \\
&\quad S_{\sigma,\overline{\delta}} \bullet S_{\sigma,\overline{\delta}} \cup S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\delta} \\
(d)\quad S_{\overline{\sigma},\overline{\delta}} &= S_{\overline{\sigma},\delta} /\!\!/ S_{\overline{\sigma},\overline{\delta}} \cup S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\sigma,\delta} \cup S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\sigma,\overline{\delta}} \cup \\
&\quad S_{\sigma,\delta} /\!\!/ S_{\overline{\sigma},\overline{\delta}} \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\overline{\sigma},\overline{\delta}} \cup S_{\sigma,\overline{\delta}} /\!\!/ S_{\overline{\sigma},\overline{\delta}} \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\sigma,\overline{\delta}} \cup \\
&\quad S_{\sigma,\overline{\delta}} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\overline{\delta}} \bullet S_{\sigma,\delta} \cup S_{\overline{\sigma},\overline{\delta}} \bullet S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\overline{\delta}} \bullet S_{\sigma,\overline{\delta}} \cup \\
&\quad S_{\overline{\sigma},\overline{\delta}} \bullet S_{\overline{\sigma},\overline{\delta}} \cup S_{\sigma,\delta} \bullet S_{\overline{\sigma},\overline{\delta}} \cup S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\overline{\delta}} \cup S_{\sigma,\overline{\delta}} \bullet S_{\overline{\sigma},\overline{\delta}}
\end{aligned}
$$

These equations are constructed as follows. Since $e_{\mathbb{Q}} = (\overline{\sigma}, \delta)$, the constant symbol e is put in the righthand side of the equation that defines $S_{\overline{\sigma},\delta}$ and nowhere else. Moreover, for $f \in \{/\!\!/, \bullet\}$, if $f_{\mathbb{Q}}(q_1, q_2) = q$, then we put the monomial $f(S_{q_1}, S_{q_2})$ in the righthand side of the equation that defines S_q. Thus, $S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\delta}$ is in the righthand side of Equation (c).

Since we have e in the righthand side of Equation (b), we have $S_{\overline{\sigma},\delta} \neq \emptyset$. Using this fact and since we have the term $S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\delta}$ in the righthand side of Equation (c), we have $S_{\sigma,\overline{\delta}} \neq \emptyset$. And by these facts and since we have $S_{\overline{\sigma},\delta} /\!\!/ S_{\sigma,\overline{\delta}}$ in the righthand side of Equation (d), we have $S_{\overline{\sigma},\overline{\delta}} \neq \emptyset$. Since every term in the righthand side of Equation (a) contains $S_{\sigma,\delta}$, we have $S_{\sigma,\delta} = \emptyset$. This proves that no series-parallel graph has one coloring of type σ and another one of type δ. Moreover, according to the proof of Proposition 1.6, this property is provable by fixed-point induction, as the reader can easily check. Using this and the commutativity of $/\!\!/$, we can simplify the system into the following one:

$$
\begin{aligned}
(b')\quad S_{\overline{\sigma},\delta} &= e \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\sigma,\overline{\delta}} \bullet S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\delta} \bullet S_{\sigma,\overline{\delta}} \\
(c')\quad S_{\sigma,\overline{\delta}} &= S_{\sigma,\overline{\delta}} /\!\!/ S_{\sigma,\overline{\delta}} \cup S_{\sigma,\overline{\delta}} \bullet S_{\sigma,\overline{\delta}} \cup S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\delta} \\
(d')\quad S_{\overline{\sigma},\overline{\delta}} &= S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\overline{\sigma},\overline{\delta}} \cup S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\overline{\delta}} /\!\!/ S_{\sigma,\overline{\delta}} \cup S_{\overline{\sigma},\delta} /\!\!/ S_{\sigma,\overline{\delta}} \cup \\
&\quad S_{\overline{\sigma},\overline{\delta}} \bullet S_{\overline{\sigma},\delta} \cup S_{\overline{\sigma},\overline{\delta}} \bullet S_{\sigma,\overline{\delta}} \cup S_{\overline{\sigma},\overline{\delta}} \bullet S_{\overline{\sigma},\overline{\delta}} \cup \\
&\quad S_{\overline{\sigma},\delta} \bullet S_{\overline{\sigma},\overline{\delta}} \cup S_{\sigma,\overline{\delta}} \bullet S_{\overline{\sigma},\overline{\delta}}
\end{aligned}
$$

Thus, this construction proves that every series-parallel graph either has no 2-coloring (it is then generated by $S_{\overline{\sigma},\overline{\delta}}$) or has one of type σ and none of type δ (it is generated by $S_{\sigma,\overline{\delta}}$) or has one of type δ and none of type σ (it is generated by $S_{\overline{\sigma},\delta}$). Let us for clarity replace $S_{\sigma,\overline{\delta}}$ by T_σ and $S_{\overline{\sigma},\delta}$ by T_δ. Then the set T of 2-colorable series-parallel graphs is defined by the equation system:

$$\begin{cases} T &= T_\sigma \cup T_\delta \\ T_\sigma &= T_\sigma /\!\!/ T_\sigma \ \cup \ T_\sigma \bullet T_\sigma \ \cup \ T_\delta \bullet T_\delta \\ T_\delta &= e \ \cup \ T_\delta /\!\!/ T_\delta \ \cup \ T_\sigma \bullet T_\delta \ \cup \ T_\delta \bullet T_\sigma \end{cases}$$

The construction of this system is based on Properties (1.10). A similar construction can be done for every equation system and every finite inductive set of properties, which proves the Filtering Theorem (Theorem 1.8).

1.3 Monadic second-order logic

We now introduce *monadic second-order logic*, a logical language with which we will specify finite inductive sets of properties. It is actually a favorite language among logicians because it is decidable for many sets of (finite or infinite) structures. Furthermore, it is suitable for expressing numerous graph properties.

1.3.1 Monadic second-order graph properties

We first explain how a graph can be made into a logical structure, hence can be a model of a sentence. For every graph G, we let $\lfloor G \rfloor$ be the relational structure[12,13] $\langle V_G, edg_G \rangle$ with domain V_G, the set of vertices. Its second component is the binary relation $edg_G \subseteq V_G \times V_G$, such that $(x, y) \in edg_G$ if and only if there exists an edge from x to y if G is directed, and an edge between x and y if G is undirected.

The classical undirected graphs K_n and $K_{n,m}$ are represented by the following relational structures[14]:

$$\begin{array}{rcl} \lfloor K_n \rfloor & := & \langle [n], edg_n \rangle, \\ edg_n(x,y) & :\Longleftrightarrow & x, y \in [n] \text{ and } \ x \neq y, \\ \lfloor K_{n,m} \rfloor & := & \langle [n+m], edg_{n,m} \rangle \\ edg_{n,m}(x,y) & :\Longleftrightarrow & 1 \leq x \leq n \text{ and } n+1 \leq y \leq n+m, \text{ or} \\ & & 1 \leq y \leq n \text{ and } n+1 \leq x \leq n+m. \end{array}$$

Properties of a graph G can be expressed by *sentences*[15] of relevant logical languages, that are interpreted in $\lfloor G \rfloor$. For example, if G is a directed graph,

[12] In some cases, we will write G instead of $\lfloor G \rfloor$.

[13] Relational structures are first-order logical structures without functions of positive arity. See Section 1.9 and Chapter 5 for detailed definitions.

[14] $[n]$ denotes $\{1, \ldots, n\}$.

[15] A sentence is a formula without free variables. The notation $S \models \varphi$ means that a sentence φ is true in the relational structure S; in that case S is said to be a model of φ.

then

$$\lfloor G \rfloor \models \forall x \exists y, z (edg(y, x) \ \wedge \ edg(x, z))$$

if and only if every vertex of G has at least one incoming edge and at least one outgoing edge (we may have $y = z = x$). If G is a simple undirected graph, then we have:

$$\lfloor G \rfloor \models \forall x \Big(\neg edg(x, x) \Big) \ \wedge \ \neg \exists w, x, y, z \Big(edg(w, x) \wedge edg(x, y) \wedge edg(y, z) \\ \wedge \neg edg(w, y) \wedge \neg edg(w, z) \wedge \neg edg(x, z) \Big)$$

if and only if G has no loop and no induced subgraph isomorphic to P_4 (P_4 is the graph $\bullet - \bullet - \bullet - \bullet$). If G is assumed finite and nonempty, this property expresses that it is a cograph. This is an alternative characterization of cographs that has no immediate relation with the grammatical definition given in Section 1.1.2.

A simple graph G is completely defined by the relational structure $\lfloor G \rfloor$: we will say that the representation of G by $\lfloor G \rfloor$ is *faithful*. This representation is not faithful for graphs with multiple edges: the graphs e and $e /\!\!/ e$ (we use here the notation of Section 1.1.3) are not the same but the structures $\lfloor e \rfloor$ and $\lfloor e /\!\!/ e \rfloor$ are. The graph properties expressed by logical formulas via the structures $\lfloor G \rfloor$ are necessarily independent of the multiplicity of edges. We will present in Section 1.8 a representation of a graph G by a relational structure denoted $\lceil G \rceil$ that is faithful, where each edge of G is also an element of the domain of $\lceil G \rceil$. The incidence between edges and vertices is represented by two binary relations in $\lceil G \rceil$ if G is directed and by only one if G is undirected. By using this alternative representation, we will be able to express properties that distinguish multiple edges.

The above two examples use first-order formulas whose variables denote vertices. Monadic second-order formulas have a richer syntax and wider expressive power. They also use variables denoting sets of vertices. Uppercase variables will denote sets of vertices, and lowercase variables will denote individual vertices. The property:

$$\lfloor G \rfloor \models \exists X \Big(\exists x. \, x \in X \ \wedge \ \exists y. \, y \notin X \ \wedge \ \forall x, y (edg(x, y) \Rightarrow (x \in X \Leftrightarrow y \in X)) \Big)$$

holds if and only if G is not connected. (We consider the empty graph as connected.) In this sentence, X is a set variable. Let γ_3 be the sentence:

$$\exists X, Y, Z \Big(Part(X, Y, Z) \wedge \\ \forall x, y \Big(edg(x, y) \ \wedge \ x \neq y \ \Rightarrow \ \neg \big(x \in X \ \wedge \ y \in X \big) \wedge \\ \neg \big(x \in Y \ \wedge \ y \in Y \big) \ \wedge \ \neg \big(x \in Z \ \wedge \ y \in Z \big) \Big) \Big)$$

where $Part(X, Y, Z)$ expresses that (X, Y, Z) is a partition[16] of the domain. The

[16] A partition may have empty components.

formula $Part(X, Y, Z)$ is written as follows:

$$\forall x\Big(\Big(x \in X \vee x \in Y \vee x \in Z\Big) \ \wedge \ \Big(\neg(x \in X \ \wedge \ x \in Y) \wedge \\ \neg(x \in Y \ \wedge \ x \in Z) \ \wedge \ \neg(x \in X \ \wedge \ x \in Z)\Big)\Big).$$

Then $\lfloor G \rfloor \models \gamma_3$ if and only if G is 3-colorable. That the ends of an edge are in different sets X, Y, Z means that they have different colors.

For each integer k, one can construct a similar sentence γ_k such that, for every graph G, we have $\lfloor G \rfloor \models \gamma_k$ if and only if G is k-colorable.

Many graph constructions can be expressed in terms of basic ones like choosing subsets and computing transitive closures of binary relations. The example of 3-colorability illustrates the first of these basic constructions. We have given a sentence expressing non-connectivity. Its negation expresses connectivity, hence, a property of the transitive closure of the relation $edg_G \cup edg_G^{-1}$. We now give an explicit construction of the transitive closure of an arbitrary binary relation.

Let R be a binary relation that is either a relation of the considered relational structure ($\lfloor G \rfloor$ or a more general one) or is defined by a formula $R(u, v)$ with free variables u and v. We say that a set X is *R-closed* if it satisfies the condition $\forall u, v\big(u \in X \wedge R(u, v) \Rightarrow v \in X\big)$. The formula $\varphi(x, y)$, defined as

$$\forall X(x \in X \wedge \text{“}X \text{ is } R\text{-closed”} \Rightarrow y \in X)$$

(where “X is R-closed” is to be replaced by the formula expressing this condition), expresses that (x, y) belongs to R^*, the *reflexive and transitive closure* of R, i.e., that there exists a finite sequence $z_1, \ldots, z_n$, such that $x = z_1$, $y = z_n$ and $(z_i, z_{i+1}) \in R$ for all $i = 1, \ldots, n-1$. We sketch the proof of this claim. If $x = z_1$, $y = z_n$ with $(z_i, z_{i+1}) \in R$ for all i, then for every R-closed set X such that x belongs to X, we have $z_i \in X$ for all $i = 1, \ldots, n$, hence $y \in X$ and $\varphi(x, y)$ holds. Conversely, if $\varphi(x, y)$ holds then one takes $X = \{z \mid (x, z) \in R^*\}$. It is R-closed, hence $y \in X$ and (x, y) belongs to R^*.

In order to have a uniform notation, we denote by $\mathrm{TC}[R; x, y]$ this formula $\varphi(x, y)$. We can use it to build a formula with free variable Y expressing that $G[Y]$, the induced subgraph of G with set of vertices Y, is connected. We let $\mathrm{CONN}(Y)$ be the formula

$$\forall x, y\big(x \in Y \wedge y \in Y \Rightarrow \mathrm{TC}[R; x, y]\big)$$

where R is the relation defined by the formula φ_R with free variables u, v and Y:

$$u \in Y \wedge v \in Y \wedge \big(edg(u, v) \vee edg(v, u)\big).$$

The variable Y is free in φ_R, hence it is also free in the monadic second-order formula $\mathrm{TC}[R; x, y]$. It is clear that the formula $\mathrm{CONN}(Y)$ expresses the desired property. This formula can be used for expressing further properties. The following sentence expresses that an undirected graph G has a cycle with at least 3 vertices:

$$\exists x, y, z\Big(x \neq y \wedge y \neq z \wedge x \neq z \wedge edg(x, z) \wedge edg(z, y) \ \wedge \\ \exists Y\big(z \notin Y \wedge x \in Y \wedge y \in Y \wedge \mathrm{CONN}(Y)\big)\Big).$$

Together with the expressibility of connectivity, we can thus express that a simple undirected graph is a tree[17].

Aiming at the expression of planarity, we examine the monadic second-order expressibility of minor inclusion. We consider undirected graphs. We say that H is a *minor* of G, denoted by $H \trianglelefteq G$ if and only if H is obtained from a subgraph G' of G by edge contractions. A graph G is planar if and only if it has no minor isomorphic to K_5 or to $K_{3,3}$. (This is a variant due to Wagner of a well-known result by Kuratowski; it is proved in the books [*Die] and [*MohaTho]).

Lemma 1.13 Let H be a simple, loop-free, undirected graph with set of vertices $[n]$. A graph G contains a minor isomorphic to H if and only if there are in G pairwise disjoint nonempty sets of vertices $Y_1, \ldots, Y_n$ such that each graph $G[Y_i]$ is connected and for every edge of H between i and j, there exists an edge in G between u and v such that $u \in Y_i$ and $v \in Y_j$. □

Corollary 1.14 For every graph H as in Lemma 1.13, there exists a monadic second-order sentence MINOR_H such that, for every undirected graph G, we have $\lfloor G \rfloor \models \mathrm{MINOR}_H$ if and only if G has a minor isomorphic to H.

Proof: The construction follows from Lemma 1.13. One takes for MINOR_H the sentence:

$$\exists Y_1, \ldots, Y_n \Big(\bigwedge_{1 \leq i \leq n} \big((\exists y.\, y \in Y_i) \wedge \mathrm{CONN}(Y_i) \big) \wedge \bigwedge_{1 \leq i < j \leq n} \neg \exists y \big(y \in Y_i \wedge y \in Y_j \big) \wedge \bigwedge_{(i,j) \in edg_H} \exists u, v \big(u \in Y_i \wedge v \in Y_j \wedge edg(u,v) \big) \Big).$$

■

Corollary 1.15 An undirected graph is planar if and only if it satisfies the sentence $\neg \mathrm{MINOR}_{K_5} \wedge \neg \mathrm{MINOR}_{K_{3,3}}$. □

With this collection of examples, the reader should have a good idea of how one can express graph properties in monadic second-order logic. However, not all graph properties can be expressed in this language. Here are some properties of a graph G and of subsets X, Y of its vertex set V_G that are not monadic second-order expressible.

P_1 : The cardinality $|X|$ of the set X is even,

P_2 : $|X|$ is a prime number,

P_3 : $|X| = |Y|$,

P_4 : G has a non-trivial automorphism,

[17] A tree is a nonempty connected undirected graph without cycles. This last condition implies that a tree has no loops and no multiple edges. The absence of loops is expressed by the sentence $\forall x(\neg edg(x,x))$, but the absence of multiple edges cannot be expressed by a sentence interpreted in $\lfloor G \rfloor$.

P_5 : G has a Hamiltonian cycle.

There are however some differences between these properties, and we have remedies in some cases. For Property P_1, the remedy consists in extending the language by adding a set predicate, $Even(X)$, expressing that the set X has even cardinality. All results that we will prove for monadic second-order logic hold for the extended language called *counting modulo* 2 *monadic second-order logic*. The notation C_2MS will refer to it (and MS will refer to formulas written without cardinality predicates).

Property P_5 is actually expressible by a sentence of monadic second-order logic that additionally uses quantifications on sets of edges, and the incidence relations between edges and vertices. This language is based on the representation of a graph G by the richer relational structure than $\lfloor G \rfloor$ that we will define in Section 1.8 and denote by $\lceil G \rceil$. It can be viewed as another extension of monadic second-order logic that we will denote by MS_2 where the index 2 recalls that there are two types of elements in the domain of $\lceil G \rceil$, vertices and edges. There are some significant differences between the languages MS and MS_2 but our main results presented in the next sections and their applications to the construction of fixed-parameter tractable algorithms have formulations that apply to MS_2 as well as to MS.

Concerning the other three properties, there is nothing to do. Adding new set predicates, say $Card_{Prime}(X)$ expressing that $|X|$ is a prime number, or $EqCard(X, Y)$ expressing that $|X| = |Y|$, or $Auto(X)$ expressing that $G[X]$ has a non-trivial automorphism, yields extensions of monadic second-order logic for which the results to be presented in Sections 1.4, 1.5 and 1.6 fail.[18]

1.3.2 Monadic second-order logic and recognizability

Logical sentences express properties of relational structures of the appropriate type. They can also be viewed as finite specifications of sets of structures, namely, their sets of models. We first make precise the corresponding terminology. For a logical language $\mathcal{L}$ (such as MS, C_2MS, or MS_2), we say that a property of relational structures over a fixed finite set of relation symbols is $\mathcal{L}$-*expressible* if it can be expressed by a sentence of $\mathcal{L}$. A set L of such structures is $\mathcal{L}$-*definable* if the membership of a structure in L is $\mathcal{L}$-expressible. These definitions are applicable to graphs represented by relational structures. Hence, with respect to a fixed representation, we will say that a graph property is $\mathcal{L}$-expressible. Examples have been given above. Let $\mathcal{C}$ be a set of graphs; an element of $\mathcal{C}$ will be called a $\mathcal{C}$-graph. We will say that a set of graphs $L \subseteq \mathcal{C}$ is an $\mathcal{L}$-*definable subset of* $\mathcal{C}$ (or, an $\mathcal{L}$-definable set of $\mathcal{C}$-graphs) if the membership of a graph in L is $\mathcal{L}$-expressible *and the considered representation is faithful for* $\mathcal{C}$*-graphs*. Hence, the connectedness of a graph G is MS-expressible with respect to its representation by $\lfloor G \rfloor$, but the set of connected graphs is not an MS-definable set of graphs, because this representation is not faithful

[18] We have actually no proof of this failure for $Card_{Prime}(X)$. We will give the proof for another numerical predicate (see the end of Section 7.5).

for graphs with multiple edges. On the other hand, the set of connected simple graphs is an MS-definable set of simple graphs. In the first case $\mathcal{C}$ is the set of all graphs, and in the second case it is the set of all simple graphs.

Let F be a finite signature. There is a bijection between $T(F)$ and a set of labelled trees that are simple graphs. It follows that every term t in $T(F)$ can be faithfully represented by a relational structure $\lfloor t \rfloor$ over a finite set of relations (the binary edge relation of the tree, and a unary relation for each label). We say that a set $L \subseteq T(F)$ is *MS-definable* if there exists an MS sentence φ such that $L = \{t \in T(F) \mid \lfloor t \rfloor \models \varphi\}$. Since the property that a finite relational structure is isomorphic to $\lfloor t \rfloor$ for some term t in $T(F)$ is itself MS-expressible, the set $L \subseteq T(F)$ is MS-definable[19] if and only if the set of relational structures that are isomorphic to some structure $\lfloor t \rfloor$ for t in L is MS-definable.

The following fundamental theorem is due to Doner [Don] and to Thatcher and Wright [ThaWri] (see Section 1.10 for related references). We will prove it in Chapters 5 and 6.

Theorem 1.16 A set of terms over a finite signature is MS-definable if and only if it is recognizable, i.e., accepted by a finite automaton. □

For the two graph algebras $\mathbb{G}^{\mathrm{u}} := \langle \mathcal{G}^{\mathrm{u}}, \oplus, \otimes, \mathbf{1} \rangle$ and $\mathbb{J}_2^{\mathrm{d}} := \langle \mathcal{J}_2^{\mathrm{d}}, /\!\!/, \bullet, e \rangle$ whose operations are defined respectively in Sections 1.1.2 and 1.1.3, we have the following results:

Proposition 1.17 Every MS-definable subset of $\mathcal{G}^{\mathrm{u}}$ is recognizable in $\mathbb{G}^{\mathrm{u}}$. Every MS_2-definable subset of $\mathcal{J}_2^{\mathrm{d}}$ is recognizable in $\mathbb{J}_2^{\mathrm{d}}$. □

This proposition is a corollary of the Recognizability Theorem (stated below in Section 1.4.3) that applies to algebras that extend $\mathbb{G}^{\mathrm{u}}$ and $\mathbb{J}_2^{\mathrm{d}}$.

1.4 Two graph algebras

Up to now, we have only given two examples of graph algebras, the algebra $\mathbb{G}^{\mathrm{u}}$ in which we have defined the cographs and the algebra $\mathbb{J}_2^{\mathrm{d}}$ in which we defined the series-parallel graphs. These algebras are subalgebras of two larger algebras that we now define. They will differ by the way in which graphs will be composed: the first algebra has operations that "bridge" two disjoint graphs by creating edges between them (vertex labels determine how these edges are created) and the second one has operations that "glue" two disjoint graphs by fusing certain vertices specified by labels. The operations from which cographs and series-parallel graphs are defined illustrate these two types of graph composition.

[19] Using MS_2 or $\mathrm{C}_2\mathrm{MS}$ sentences for defining sets of terms would not yield a wider class of definable sets.

1.4.1 The algebra of simple graphs with ports

Our first graph algebra, called the *VR algebra*[20], generalizes the algebra $\mathbb{G}^{\mathrm{u}}$. In order to define more powerful edge creating operations than $\otimes$, we will use vertex-labelled graphs. We let $\mathcal{A}$ be a countable set of labels. In this overview chapter, we take it equal to $\mathcal{N}$, the set of nonnegative integers. This will simplify some statements; in Chapter 2 we will assume that $\mathcal{N} \subseteq \mathcal{A}$. We let $\mathcal{G}$ be the set of (abstract) simple directed graphs[21]. This set contains $\mathcal{G}^{\mathrm{u}}$ because for simple graphs we consider an undirected edge as a pair of directed opposite edges. This is coherent with the representation of a graph G by $\lfloor G \rfloor$ defined in Section 1.3.1. We let $\mathcal{GP}$ be the set of (abstract) *graphs with ports*, or *p-graphs* in short, defined as pairs $G = \langle G^\circ, \mathit{port}_G \rangle$ where $G^\circ \in \mathcal{G}$ and port_G is a mapping $V_{G^\circ} \to \mathcal{A}$. If $\mathit{port}_G(u) = a \in \mathcal{A}$ we say that u is an *a-port* of G and that a is its *port label*. Every graph in $\mathcal{G}$ will be considered as the p-graph, all vertices of which are 1-ports; hence $\mathcal{G} \subseteq \mathcal{GP}$. The operations on $\mathcal{GP}$ are the following ones. First, the disjoint union[22]:

$$G \oplus H := \langle G^\circ \oplus H^\circ,\ \mathit{port}_G \cup \mathit{port}_H \rangle.$$

Then we define unary operations that manipulate port labels. For $a, b \in \mathcal{A}$ and $G = \langle G^\circ, \mathit{port}_G \rangle$ we let:

$$\begin{aligned} &\mathit{relab}_{a\to b}(G) := \langle G^\circ, \mathit{port} \rangle \text{ where} \\ &\mathit{port}(u) := \texttt{if } \mathit{port}_G(u) = a \texttt{ then } b \texttt{ else } \mathit{port}_G(u). \end{aligned}$$

The next unary operations add directed edges.[23] For $a \neq b$, we define

$$\begin{aligned} &\overrightarrow{\mathit{add}}_{a,b}(G) \text{ as } \langle G', \mathit{port}_G \rangle \text{ where } V_{G'} = V_G \text{ and} \\ &\mathit{edg}_{G'} \text{ is } \mathit{edg}_{G^\circ} \cup \{(u,v) \mid (\mathit{port}_G(u), \mathit{port}_G(v)) = (a,b)\}. \end{aligned}$$

This operation adds an edge linking u to v whenever u is an a-port and v is a b-port, unless there already exists one (we only consider simple graphs). It does not add loops.

We let $\mathbf{1}$ be a constant symbol denoting a single vertex that is a 1-port and we let $\mathbf{1}^\ell$ denote the same graph with a loop. The only way to define loops is by means of these constant symbols. Moreover, we let $\varnothing$ be a constant symbol denoting the empty graph, that we denote also by $\varnothing$. We denote empty sets[24] by the different symbol $\emptyset$.

[20] We call it in this way because its equational sets, the VR-*equational sets of graphs*, are the sets of graphs generated by certain context-free graph grammars whose rewritings are based on *Vertex Replacement*. See [*EngRoz] or [*Eng97] for comprehensive surveys.

[21] "Abstract" means that two isomorphic graphs are considered as equal. This notion will be formalized in Chapter 2.

[22] We assume G and H disjoint. If they are not, for instance if $H = G$, we replace H by an isomorphic copy disjoint from G. It follows that $\oplus$ is a well-defined binary function on isomorphism classes of graphs, i.e., on abstract graphs.

[23] To add undirected edges, we use $\overrightarrow{\mathit{add}}_{b,a}(\overrightarrow{\mathit{add}}_{a,b}(G))$. We will denote by $\mathit{add}_{a,b}$ the unary operation that transforms G into $\overrightarrow{\mathit{add}}_{b,a}(\overrightarrow{\mathit{add}}_{a,b}(G))$.

[24] We consider that an empty set of numbers is not equal to an empty set of words.

We obtain the algebra $\mathbb{GP}$ of (abstract) p-graphs, also called the *VR algebra*. Its domain is $\mathcal{GP}$ and its signature, denoted by F^{VR}, is the countable set of operations introduced above. For every term $t \in T(F^{\mathrm{VR}})$, we denote by $t_{\mathbb{GP}}$ its value, which is a p-graph, computed according to the definitions of the operations of F^{VR}. The equational sets of $\mathbb{GP}$ are called the VR-*equational sets*.

The complete join $G \otimes H$ can be defined in terms of the operations of F^{VR} as follows:

$$G \otimes H := relab_{2\to 1}(\overrightarrow{add}_{2,1}(\overrightarrow{add}_{1,2}(G \oplus relab_{1\to 2}(H)))),$$

hence by the term $relab_{2\to 1}(\overrightarrow{add}_{2,1}(\overrightarrow{add}_{1,2}(x_1 \oplus relab_{1\to 2}(x_2))))$. Such an operation, defined by a term with variables, is called a *derived operation* of the relevant algebra, here $\mathbb{GP}$.[25] Note that although this operation uses the port label 2, it transforms two graphs G, H in $\mathcal{G}$ ($\mathcal{G}$ is the set of graphs, all vertices of which are 1-ports) into a graph in the same set.

Systems of equations (that define VR-equational sets) can frequently be written more clearly with the help of derived operations. For example, Equation (1.1) in Section 1.1.2 can be written as the following equation:

$$C = (C \oplus C) \cup relab_{2\to 1}(\overrightarrow{add}_{2,1}(\overrightarrow{add}_{1,2}(C \oplus relab_{1\to 2}(C)))) \cup \mathbf{1},$$

where C defines a subset of $\mathcal{GP}$, but it is more readable as in (1.1).

For each $k \in \mathcal{N}$ we denote by $F^{\mathrm{VR}}_{[k]}$ the finite subsignature of F^{VR} consisting of the operations written with port labels in $[k]$.[26] Hence,

$$F^{\mathrm{VR}}_{[k]} := \{\oplus, relab_{a\to b}, \overrightarrow{add}_{a,b}, \mathbf{1}, \mathbf{1}^{\ell}, \varnothing \mid 1 \le a, b \le k\}.$$

It is not hard to see that every p-graph with n vertices and port labels in $[n]$ is the value of a term t in $F^{\mathrm{VR}}_{[n]}$. However, in many cases, much fewer labels suffice, and for algorithmic applications it is useful to use as few labels as possible.

In this perspective, we define the *clique-width* of a graph G, either directed or undirected, as the minimum k such that G is the value of a term in $T(F^{\mathrm{VR}}_{[k]})$. This number is denoted by $cwd(G)$. Trees have clique-width at most 3. Cographs have clique-width at most 2: this is clear because the above equation that defines cographs uses only two port labels.

Proposition 1.18 Every VR-equational set of graphs has bounded clique-width. For each k, the set of graphs of clique-width at most k is VR-equational.

Proof sketch: Let L be a VR-equational set of graphs. It is defined by an equation system written with port labels in some set $[k]$. It follows that all p-graphs belonging to the components of its least solution are the values of terms in $F^{\mathrm{VR}}_{[k]}$. In particular, the graphs in L have clique-width at most k.

[25] The unary operation $add_{a,b}$ is also a derived operation of $\mathbb{GP}$, defined by the term $\overrightarrow{add}_{b,a}(\overrightarrow{add}_{a,b}(x_1))$. The operations r of the P-algebra $\mathbb{M}'$ in Section 1.1.5 are derived operations of the F-algebra $\mathbb{M}$, defined by terms t_r.

[26] Recall that $[k] = \{1, \ldots, k\}$ for $k \in \mathcal{N}$ with $[k] = \emptyset$.

Conversely consider the sets of p-graphs S and T defined by the equation system:

$$\left\{\begin{array}{lcl} S & = & (S \oplus S) \cup \bigcup f(S) \cup \mathbf{1} \cup \mathbf{1}^{\ell} \cup \varnothing \\ T & = & relab_{2\to 1}(\cdots(relab_{k\to 1}(S)\cdots)) \end{array}\right. \tag{1.11}$$

where the union extends to all unary operations f belonging to $F^{\mathrm{VR}}_{[k]}$. In the equation that defines T, all port labels are transformed into 1. The set S consists of all p-graphs defined by terms in $T(F^{\mathrm{VR}}_{[k]})$ and the set T consists of those whose vertices are all 1-ports, hence of all graphs of clique-width at most k. ■

At the cost of some technicalities that we want to avoid here, the notion of clique-width and Proposition 1.18 can be extended to graphs with ports. However, port labels are just a tool to construct graphs. Hence Proposition 1.18 states the main facts about the relationship between VR-equational sets of graphs and clique-width.

The Clique-Width Checking problem consists in deciding whether or not $cwd(G) \leq k$ for given (G, k). It is NP-complete ([FelRRS]). It is not known whether this problem is polynomial for fixed k, when $k \geq 4$, but it is when $k \leq 3$ ([CorHLRR]).

1.4.2 The algebra of graphs with sources

We now define an algebra of graphs with multiple edges, called the *HR algebra*[27], that extends the algebra $\mathbb{J}_2^{\mathrm{d}} = \langle \mathcal{J}_2^{\mathrm{d}}, /\!\!/, \bullet, e\rangle$ considered in Section 1.1.3.

We consider (abstract) directed or undirected graphs, possibly with multiple edges. They form the set $\mathcal{J}$. For a graph in $\mathcal{J}$, E_G denotes its set of edges (and V_G its set of vertices). We let $\mathcal{A}$ be a countable set of labels (as in Section 1.4.1 we take it equal to $\mathcal{N}$) that will be used to distinguish particular vertices. These distinguished vertices will be called *sources*, and $\mathcal{A}$ is the set of *source labels*. (This notion of source is unrelated with edge directions).

A *graph with sources*, or *s-graph* in short, is a pair $G = \langle G^{\circ}, src_G\rangle$ where $G^{\circ} \in \mathcal{J}$ and src_G is a bijection from a finite subset $\tau(G)$ of $\mathcal{A}$ to a subset of $V_{G^{\circ}}$. We call $\tau(G)$ the *type* of G and $src_G(\tau(G))$ the set of its sources. The vertex $src_G(a)$ is called the *a-source* of G; its *source label*, also called its *source name*, is a.

We let $\mathcal{JS}$ denote the set of s-graphs; $\mathcal{J}$ is thus the set of s-graphs having an empty type. Clearly, $\mathcal{J}_2^{\mathrm{d}} \subseteq \mathcal{JS}$ and the elements of $\mathcal{J}_2^{\mathrm{d}}$ are s-graphs of type $\{1, 2\}$. We define operations on $\mathcal{JS}$: first a binary operation called the *parallel-composition*, a particular case of which has been defined in Section 1.1.3. For $G, H \in \mathcal{JS}$ we let

$$G /\!\!/ H := \langle G \cup H',\ src_G \cup src_{H'}\rangle$$

[27] We call it in this way because its equational sets, the HR-*equational sets of graphs*, are the sets of graphs generated by certain context-free graph grammars whose rewritings are based on *Hyperedge Replacement*. We will define them in Chapter 4, Section 4.1.5. For a thorough study, see [*DreKH] or [*Hab].

where H' is isomorphic to H[28] and is such that

$$E_{H'} \cap E_G = \emptyset,$$
$$src_G(a) = src_{H'}(a) \text{ if } a \in \tau(G) \cap \tau(H'),$$
$$V_G \cap V_{H'} = \{src_G(a) \mid a \in \tau(G) \cap \tau(H')\}.$$

This operation "glues" G and a disjoint copy of H by fusing their sources having the same names. The s-graph $G /\!\!/ H$ is well defined up to isomorphism. Its type is $\tau(G) \cup \tau(H)$.

We define unary operations that manipulate source labels. Let $G = \langle G^\circ, src_G \rangle$ be an s-graph. For $a \in \mathcal{A}$, we let $fg_a(G) := \langle G^\circ, src' \rangle$ where $src'(b)$ is $src_G(b)$ if $b \in \tau(G) - \{a\}$ and is undefined otherwise. We say that fg_a *forgets* the a-source: if G has an a-source, then this vertex is no longer distinguished as a source in $fg_a(G)$ and is turned into an "ordinary", we will say *internal*, vertex. Hence $\tau(fg_a(G)) = \tau(G) - \{a\}$.

The next operation modifies source names; it is called a *renaming*. For $a, b \in \mathcal{A}$, $a \neq b$, we let

$$\begin{array}{lcll} ren_{a \leftrightarrow b}(G) & := & \langle G^\circ, src' \rangle \text{ where} & \\ src'(a) & := & src_G(b) & \text{if } b \in \tau(G), \\ src'(b) & := & src_G(a) & \text{if } a \in \tau(G), \\ src'(c) & := & src_G(c) & \text{if } c \in \tau(G) - \{a, b\}, \end{array}$$

and src' is undefined otherwise. Hence

$$\tau(ren_{a \leftrightarrow b}(G)) = \begin{cases} \tau(G) & \text{if } a, b \in \tau(G) \text{ or } \tau(G) \cap \{a, b\} = \emptyset, \\ (\tau(G) - \{a\}) \cup \{b\} & \text{if } a \in \tau(G),\ b \notin \tau(G), \\ (\tau(G) - \{b\}) \cup \{a\} & \text{if } b \in \tau(G),\ a \notin \tau(G). \end{cases}$$

We can write this more succinctly as $\tau(ren_{a \leftrightarrow b}(G)) = \tau(G)[b/a, a/b]$ where, for every set C, we denote by $C[c/a, d/b]$ the result of the simultaneous replacement in C of a by c and of b by d.

We also define constant symbols: $\mathbf{ab}$, $\overrightarrow{\mathbf{ab}}$, $\mathbf{a}$, $\mathbf{a}^\ell$, $\varnothing$ to denote respectively an undirected edge linking an a-source and a b-source, a directed edge from an a-source to a b-source, a single vertex that is an a-source, an a-source with a loop, and the empty graph. (Since we can change source names, it would suffice to use the constant symbols $\mathbf{12}$, $\overrightarrow{\mathbf{12}}$, $\mathbf{1}$, $\mathbf{1}^\ell$, $\varnothing$. However, using many renaming operations would make terms denoting graphs unreadable.)

We obtain a countable signature denoted by F^{HR} and an F^{HR}-algebra of graphs denoted by $\mathbb{JS}$ and called the *HR algebra*. It has domain $\mathcal{JS}$. As for $\mathbb{GP}$, for every term $t \in T(F^{\mathrm{HR}})$, we denote by $t_{\mathbb{JS}}$ the s-graph that is its value. The equational sets of $\mathbb{JS}$ are called the HR-*equational sets*.

As for VR-equational sets, the equations that define HR-equational sets can be shortened if they are written with derived operations. For example, the

[28] H' isomorphic to H implies that $\tau(H') = \tau(H)$.

series-composition of graphs of type $\{1,2\}$ is a derived operation that can be expressed by:

$$G \bullet H := fg_3(ren_{2\leftrightarrow 3}(G) \mathbin{/\!\!/} ren_{1\leftrightarrow 3}(H)).$$

We denote by $F^{\mathrm{HR}}_{[k]}$ the finite subsignature of F^{HR} consisting of the operations $/\!\!/$, fg_a, $ren_{a\leftrightarrow b}$, for $a, b \in [k]$ and of the constant symbols denoting graphs with source names in $[k]$.

Each s-graph G with n vertices and source names in $[n]$ is the value of a term in $T(F^{\mathrm{HR}}_{[n]})$. The least integer k such that a graph G (without sources) is the value of a term in $T(F^{\mathrm{HR}}_{[k+1]})$ is a well-known graph complexity measure called the *tree-width* of G and denoted by $twd(G)$. It has been defined previously in a combinatorial way by Robertson and Seymour and by other authors using a different terminology. In the combinatorial definition, $twd(G)$ is the least integer k such that G has a so-called *tree-decomposition* of *width* k. A tree-decomposition of G is a decomposition of G into a tree of subgraphs, where each node of the tree corresponds to a subgraph of G; its width is the maximal number of vertices of those subgraphs, minus 1. The notion of tree-width is important for the construction of graph algorithms and for the study of graph minors: see the books [*Die], [*DowFel] and [*FluGro], and the survey articles [*Bod93] and [*Bod98]. We will study this combinatorial notion in Chapter 2 and prove the following result which is an algebraic characterization of it:

Proposition 1.19 A graph has tree-width at most k if and only if it is the value of a term in $T(F^{\mathrm{HR}}_{[k+1]})$. □

By contrast clique-width, defined in terms of graph operations, has yet no alternative combinatorial definition. Using Proposition 1.19 the following result can be proved in the same way as Proposition 1.18.

Proposition 1.20 Every HR-equational set of graphs has bounded tree-width. For each k, the set of graphs of tree-width at most k is HR-equational. □

1.4.3 A weak Recognizability Theorem

The VR algebra $\mathbb{GP}$ has a countable signature F^{VR} that generates it: this means that each element is the value of some term. For each k in $\mathcal{N}$, we let $\mathbb{GP}^{\mathrm{gen}}[k]$ be the subalgebra of $\mathbb{GP}$ that is generated by the finite subsignature $F^{\mathrm{VR}}_{[k]}$ of F^{VR}. Its domain $\mathcal{GP}^{\mathrm{gen}}[k]$ consists of the graphs with ports that are values of terms in $T(F^{\mathrm{VR}}_{[k]})$. We define similarly $\mathbb{JS}^{\mathrm{gen}}[k]$ as the subalgebra of the HR algebra $\mathbb{JS}$ that is generated by the finite subsignature $F^{\mathrm{HR}}_{[k]}$ of F^{HR}. Its domain is $\mathcal{JS}^{\mathrm{gen}}[k]$. Proposition 1.17 extends into the following theorem:

Theorem 1.21 (Weak Recognizability Theorem)

(1) Let L be a CMS-definable set of simple graphs. For every $k \in \mathcal{N}$, the set $L \cap \mathcal{GP}^{\mathrm{gen}}[k]$ is recognizable in the algebra $\mathbb{GP}^{\mathrm{gen}}[k]$.

(2) Let L be an CMS_2-definable set of graphs. For every $k \in \mathcal{N}$, the set $L \cap \mathcal{JS}^{\mathrm{gen}}[k]$ is recognizable in the algebra $\mathbb{JS}^{\mathrm{gen}}[k]$. □

In this statement, CMS refers to *Counting Monadic Second-order* logic, i.e., to the use in monadic second-order formulas of set predicates $Card_p(X)$ expressing that $|X|$ is a multiple of p ($Even(X)$ is thus $Card_2(X)$). Note that $L \cap \mathcal{GP}^{\mathrm{gen}}[k]$ is the set of graphs in L that have clique-width at most k, and similarly for $L \cap \mathcal{JS}^{\mathrm{gen}}[k]$ and tree-width at most $k-1$.

This theorem is a consequence of the (more powerful) Recognizability Theorem to be stated and proved in Chapter 5. The latter theorem is more powerful in several respects. First it yields recognizability with respect to the algebras $\mathbb{GP}$ and $\mathbb{JS}$ that have infinite signatures and not only with respect to their finitely generated subalgebras $\mathbb{GP}^{\mathrm{gen}}[k]$ and $\mathbb{JS}^{\mathrm{gen}}[k]$. We postpone to Chapters 3 and 4 the detailed definitions. Second, the Recognizability Theorem can be stated and proved for an algebra of relational structures denoted by $\mathbb{STR}$, as a unique statement that entails the cases of $\mathbb{GP}$ and $\mathbb{JS}$.

However, Theorem 1.21 has already interesting consequences. We first state the logical version of the Filtering Theorem and its applications to decidability results for monadic second-order sentences. Applications to fixed-parameter tractability will be considered in the next section. We will discuss decidability results in more detail in Section 1.6.

Theorem 1.22 (Filtering Theorem, logical version)

(1) For every VR-equational set of graphs L and every CMS-expressible graph property P, the set L_P consisting of the graphs of L that satisfy P is VR-equational.

(2) The analogous result holds for HR-equational sets and CMS_2-expressible properties. □

This result is a direct consequence of Theorems 1.8, 1.12 and 1.21; note that each VR-equational set is equational in some algebra $\mathbb{GP}^{\mathrm{gen}}[k]$, cf. the proof of Proposition 1.18. All constructions are effective: an equation system defining L_P can be constructed from one defining L and a sentence expressing P. Now consider Corollary 1.9 and its proof. Since the emptiness of an equational set is decidable, one can decide if L_P is nonempty, i.e., if P is satisfied by some graph in L: the CMS-*satisfiability problem* (resp. the CMS_2-*satisfiability problem*) is decidable for VR-equational sets (resp. for HR-equational sets) and, in particular, for the sets $CWD(\leq k)$ of graphs of clique-width at most k (resp. for the sets $TWD(\leq k)$ of graphs of tree-width at most k).

We now come back to statements (1) and (2) at the end of Section 1.2.2. The set $L_{\neg P}$ (we use the notation of Theorem 1.22) is also VR-equational (respectively, HR-equational in the second case). Its emptiness can be tested. That $L_{\neg P}$ is empty means that P is universally valid on L. From the equation system defining $L_{\neg P}$, one can also obtain a proof of this fact by fixed-point induction, according to the generalization of Proposition 1.6 to recognizable sets. As observed in Section 1.2.5, if P belongs to a finite inductive set $\mathcal{P}$ of properties,

then the auxiliary properties used in that proof are Boolean combinations of the properties in $\mathcal{P}$. In Chapter 5, the proof of Theorem 1.21 shows that, more precisely, every CMS-expressible property P belongs to a finite inductive set $\mathcal{P}$ of *CMS-expressible* properties in the algebra $\mathbb{GP}^{\mathrm{gen}}[k]$ (and similarly for CMS_2 and $\mathbb{JS}^{\mathrm{gen}}[k]$). Hence, the auxiliary properties in the proof by fixed-point induction are CMS-expressible (respectively CMS_2-expressible).

1.5 Fixed-parameter tractability

In this section, we describe some algorithmic applications of the Weak Recognizability Theorem. We first recall the basic definition of *fixed-parameter tractability*. This notion has been introduced in order to describe in an abstract setting the frequently met situation where the computation time of an algorithm is bounded by an expression of the form $f(p(d)) \cdot |d|^c$, where the set of inputs is equipped with two computable integer valued functions, a *size* function $d \mapsto |d|$ (d is the generic input) and a *parameter* function $d \mapsto p(d)$, such that $0 \leq p(d) \leq |d|$, and where f is a fixed computable function of nonnegative integers and c is a fixed positive integer. If these conditions are satisfied, the considered algorithm is *fixed-parameter tractable with respect to the parameter* p. If $c = 1, 2$ or 3, we say that it is *fixed-parameter linear, quadratic* or *cubic* respectively.

As size of an input graph G, we will use either the number of its vertices and edges, denoted by $\|G\|$, or, in particular if G is simple, its number of vertices. Parameters can be the degree of G, its tree-width or its clique-width. Our results will use these last two values; other examples can be found in the books [*DowFel] and [*FluGro] which present in detail the theory of fixed-parameter tractability.

The size $|t|$ of a term t or the size $|\varphi|$ of a formula φ is, roughly speaking, the number of symbols with which t or φ is written, i.e., the length of the corresponding word.[29]

Example 1.23 The *subgraph isomorphism problem* consists in deciding for a pair of simple graphs (G, H) whether H is isomorphic to a subgraph of G. It is NP-complete (Problem GT48 in [*GarJoh]). For each fixed graph H, this problem can be solved in time $O(n^m)$ where $n = |V_G|$ and $m = |V_H|$. However it is fixed-parameter linear with respect to $Deg(G)$, the degree of G. This follows from a result by Seese [See96] (another proof is given in [DurGra]) saying that for every first-order sentence φ, one can decide in time bounded by $f(|\varphi| + Deg(G)) \cdot |V_G|$ whether $\lfloor G \rfloor$ satisfies φ, for some fixed computable function f. The property that G has a subgraph isomorphic to a fixed graph H is expressible by a first-order sentence φ_H to be interpreted in $\lfloor G \rfloor$. Hence the subgraph isomorphism problem can be solved in time at most $f(|\varphi_H| + Deg(G)) \cdot |V_G|$. □

[29] Formal definitions of $|t|$ and $|\varphi|$ will be given in Chapters 2 and 5, respectively.

Fixed-parameter tractable algorithms that check monadic second-order graph properties can be derived from Theorems 1.16 and 1.21. The *model-checking problem* for a logical language $\mathcal{L}$ and a class of structures $\mathcal{C}$ consists in deciding whether $S \models \varphi$, for a given structure $S \in \mathcal{C}$ and a given sentence $\varphi \in \mathcal{L}$.

Theorem 1.24 For every finite signature F, every monadic second-order sentence φ and every term t in $T(F)$, one can decide in time at most $f(F,\varphi) \cdot |t|$ whether $\lfloor t \rfloor \models \varphi$, where f is a fixed computable function.

Proof: The proof of Theorem 1.16 being effective, it yields an algorithm that constructs from F and φ a finite deterministic F-automaton $\mathscr{A}$ accepting the set of terms s in $T(F)$ such that $\lfloor s \rfloor \models \varphi$. By running $\mathscr{A}$ on a given term t, one gets the answer in time proportional to $|t|$. The total computation time is thus $f_1(F,\varphi) + f_2(F,\varphi) \cdot |t|$ where $f_1(F,\varphi)$ is the time taken to compute $\mathscr{A}$ and $f_2(F,\varphi)$ is the maximum time taken by $\mathscr{A}$ to perform one transition. ■

In other words, the model-checking problem for monadic second-order sentences and the class of terms is fixed-parameter linear with respect to $\|F\| + |\varphi|$, where $\|F\|$ is the sum of arities of the symbols in F plus the number of constant symbols in F. This follows from Theorem 1.24 because, up to the names of the symbols in F and of the variables in φ, there are finitely many pairs (F,φ) such that $\|F\| + |\varphi| \leq p$, so that $f(F,\varphi)$ can be bounded by $g(\|F\| + |\varphi|)$ for some computable function g.

We now consider the model-checking problem for monadic second-order sentences on graphs and its two possible parametrizations by tree-width and clique-width. For a graph G given by a term t in $T(F^{\mathrm{HR}}_{[k]})$ or in $T(F^{\mathrm{VR}}_{[k]})$, one gets a fixed-parameter linear algorithm as in Theorem 1.24 because, by Theorems 1.21 and 1.12, one can construct from k and φ a finite deterministic $F^{\mathrm{HR}}_{[k]}$-automaton $\mathscr{A}_{\varphi,k}$ that accepts the set of terms t such that $t_{\mathbb{IS}}$ satisfies φ, and similarly with $F^{\mathrm{VR}}_{[k]}$. This situation happens in particular if G belongs to an HR- or VR-equational set of graphs and is given by a corresponding term or derivation tree (cf. Section 1.1.5). However, if such a term or derivation tree is not given, it must be computed from the input graph by a parsing algorithm.

Theorem 1.25

(1) The model-checking problem for CMS sentences and the class of simple graphs is fixed-parameter cubic with respect to $cwd(G) + |\varphi|$. The input sentence is φ and the size of the input graph G is its number of vertices.

(2) The model-checking problem for CMS_2 sentences and the class of graphs is fixed-parameter linear with respect to $twd(G) + |\varphi|$. The input sentence is φ and the size of the input graph G is its number $\|G\|$ of vertices and edges.[30]

[30] In many cases, φ is fixed because one is interested in a particular graph property, and then the parameters are just tree-width and clique-width.

Proof: We first consider the parametrization by tree-width. We let n be the number of vertices and edges of the input graph G, i.e., $n := \|G\|$. There exists an algorithm (by Bodlaender [Bod96], see also [*DowFel]) that, for every graph G and integer k, decides if $twd(G) \leq k$ in time at most $g(k) \cdot |V_G|$ (for some fixed computable function g), and that constructs if possible a tree-decomposition of width k of G; this tree-decomposition can be converted in linear time (in n) into a term t in $T(F^{\mathrm{HR}}_{[k+1]})$ that evaluates to G, cf. Proposition 1.19.

Step 1: For given G and by repeating this algorithm for $k = 1, 2, \ldots$ at most $twd(G)$ times one obtains an optimal tree-decomposition of G. This computation takes time at most $twd(G) \cdot g'(twd(G)) \cdot n$ for some fixed computable function g' and builds a term in $T(F)$ where $F := F^{\mathrm{HR}}_{[twd(G)+1]}$, that evaluates to G. (The function g' takes into account the time needed to transform a tree-decomposition into a term.)

Step 2: By using Theorems 1.21 and 1.12, one constructs a finite deterministic F-automaton $\mathscr{A}_{\varphi,k}$ where $k = twd(G) + 1$ that accepts the set of terms $t \in T(F)$ that evaluate to a graph satisfying φ.

Step 3: By running this automaton on t, one obtains the answer.

We now consider the parametrization by clique-width. The main difference concerns Step 1. There exists an algorithm (that combines algorithms from [HliOum] and [OumSey], see Section 6.2.3 for details) that, for every simple graph G and every integer k, either reports (correctly) that $cwd(G) > k$ or constructs a term in $T(F^{\mathrm{VR}}_{[h(k)]})$ that evaluates to G (and hence G has clique-width at most $h(k)$) where $h(k) = 2^{k+1} - 1$. This algorithm takes time $g''(k) \cdot n^3$ for some fixed computable function g'', where $n = |V_G|$. Note that it does not determine the exact clique-width of G.

Step 1 of the case of tree-width is replaced by the following: for given G and by repeating this algorithm for $k = 1, 2, \ldots$ at most $cwd(G)$ times, one obtains a term t in $T(F^{\mathrm{VR}}_{[m]})$ that evaluates to G for some $m \leq h(cwd(G))$. This computation takes time at most $cwd(G) \cdot f(cwd(G)) \cdot n^3$, where $f(k)$ is the maximum of $g''(i)$ for $i = 1, \ldots, k$. The computation continues then by constructing a finite $F^{\mathrm{VR}}_{[m]}$-automaton (by Theorems 1.21 and 1.12) and running it on t like in Steps 2 and 3 above. ■

The algorithms of Theorem 1.25 are actually not directly implementable because of the sizes of the automata to be constructed. Specifically, the automata $\mathscr{A}_{\varphi,k}$ in the two cases of Theorem 1.25 have a number of states that is not bounded by a function of the form $\exp \circ \exp \circ \cdots \circ \exp(|\varphi|)$ with a fixed number of iterated exponentiations ($\exp(n) = 2^n$ for every n). (This is also the case for the automaton constructed in the proof of Theorem 1.24). This fact is not a weakness of the construction, but a consequence of the fact that complicated properties can be expressed by short formulas. This phenomenon occurs for first-order as well as for monadic second-order logic, as proved in [FriGro04]. Concrete constructions of $F^{\mathrm{VR}}_{[k]}$-automata for small values of k and simple graph properties will be presented in Section 6.3.

A second reason that makes these algorithms difficult, if not impossible, to implement so as to run for arbitrary graphs is the time needed for parsing the input graphs, i.e., for building terms in $T(F^{\mathrm{HR}}_{[k]})$ or in $T(F^{\mathrm{VR}}_{[k]})$ for given values of k that evaluate to them. The linear algorithm by Bodlaender [Bod96] used in the proof of Theorem 1.25 takes time $2^{32k^3} \cdot n$. We will review other more efficient, even if not linear, algorithms in Chapter 6. The cubic algorithm of [HliOum] is not implementable either.

1.6 Decidability of monadic second-order logic

Apart from model-checking discussed in the previous section, another major problem in Logic consists in deciding whether a given sentence holds in some relational structure (or in a graph represented by a relational structure) of a fixed set L. In this case, the input of the problem is an arbitrary sentence from a logical language $\mathcal{L}$. This problem is called the *$\mathcal{L}$-satisfiability problem* for the set L. A related problem consists in deciding if a given sentence of $\mathcal{L}$ belongs to the *$\mathcal{L}$-theory* of L, that is, to the set of sentences of $\mathcal{L}$ that hold for all graphs (or structures) in L. We say that the $\mathcal{L}$-theory of L is *decidable* if this problem is decidable. As logical languages $\mathcal{L}$, we will consider fragments and extensions of monadic second-order logic that are closed under negation. For such languages, the $\mathcal{L}$-satisfiability problem for a set L is decidable if and only if the $\mathcal{L}$-theory of L is decidable.

The main motivation for the fundamental Theorem 1.16 was to prove the decidability of the MS-theory of the set of terms $T(F)$ (more precisely, the set of structures $\{\lfloor t \rfloor \mid t \in T(F)\}$), for every finite signature F. From Theorem 1.22 we obtain a similar result for graphs, as observed at the end of Section 1.4.3.

Theorem 1.26 The CMS-theory of the set $CWD(\leq k)$ of simple graphs of clique-width at most k, or of a VR-equational set of graphs is decidable. So is the CMS_2-theory of the set $TWD(\leq k)$ of graphs of tree-width at most k, or of an HR-equational set of graphs. □

One obtains results which are quite powerful in that they apply to many different sets of graphs. One might hope to use them in order to obtain automatic proofs of conjectures or of difficult theorems in graph theory. However the situation is not so favorable. Let us take the example of the 4-Color Theorem, stating that every planar graph is 4-colorable. In Section 1.3.1 we have shown the existence of two MS sentences, π and γ_4, expressing respectively that a graph is planar (Corollary 1.15) and that it is 4-colorable. The 4-Color Theorem[31] can thus be stated in the following logical form:

Theorem 1.27 We have $\lfloor G \rfloor \models \pi \Rightarrow \gamma_4$ for every graph G in $\mathcal{G}$. □

[31] Its proof by Robertson *et al.* ([RobSanST]) has been checked by computer by Gonthier [Gon] with the software Coq based on Type Theory.

This means that the MS sentence $\pi \wedge \neg\gamma_4$ is not satisfiable in $\mathcal{G}$.

Certain conjectures can be formulated in a similar way. For example, a conjecture by Hadwiger states that for every integer k, if a graph is not k-colorable, then it has a minor isomorphic to K_{k+1}. For each k, the corresponding instance of this conjecture is equivalent by Corollary 1.14 to the statement:

Conjecture 1.28 We have $\lfloor G \rfloor \models \neg\gamma_k \Rightarrow \mathrm{MINOR}_{K_{k+1}}$ for every graph G in $\mathcal{G}$. □

Robertson *et al.* have proved it in [RobST] for $k = 5$. It is known to hold for smaller values and is otherwise open.

> Could one prove the 4-Color Theorem or this Conjecture for some fixed $k \geq 6$ by an algorithm able to check the satisfiability of a monadic second-order sentence?

This is not possible without some further analysis that would limit the size, or the tree-width, or the clique-width of a minimal graph G that could contradict the considered properties[32]. The reason is that the MS-satisfiability problem for the set $\mathcal{G}$ of finite (simple directed) graphs is undecidable: there is no algorithm that would take as input an arbitrary monadic second-order sentence and tell whether this sentence is valid in the logical structure $\lfloor G \rfloor$ for some graph G in $\mathcal{G}$. This undecidability result actually holds for first-order logic (see Theorem 5.5 in Section 5.1.6 or the books [*EbbFlu] and [*Lib04]).

From Theorem 1.26, it follows that the *particular cases* of these theorems or conjectures obtained by restricting to the sets of graphs of clique-width at most k for fixed values of k can, at least in principle, be proved by machine. However, since we observed that the algorithms underlying Theorem 1.26 are not implementable, this possibility is presently purely theoretical. Furthermore, the difficult open questions of graph theory concern usually all graphs rather than graphs of bounded tree-width or clique-width. There are however some exceptions. Whether the oriented chromatic number of an oriented graph[33] is equal to k is expressible by a formula of monadic second-order logic (one formula for each k). Several articles, in particular by Sopena [Sop] and Fertin *et al.* [FerRR], determine the maximal value of the oriented chromatic number of outerplanar graphs, of 3-trees, and of the so-called "fat trees" and "fat fat trees". Since these four sets of graphs are HR-equational, the maximal values of the oriented chromatic numbers of their graphs can also be determined, in principle, by algorithms based on Theorem 1.26.

Seese raised in [See91] the question of understanding which conditions on a set of graphs L are necessary for its MS-satisfiability problem to be decid-

[32] For Conjecture 1.28, Kawarabayashi [Kaw] has found such bounds for all k. It follows that each level of the conjecture is decidable, but by an intractable algorithm.

[33] An oriented graph G (i.e., a graph without loops and pairs of opposite directed edges) has *oriented chromatic number* at most k if there exists a tournament (a complete oriented graph) H with k vertices and a homomorphism $h : G \to H$ that maps V_G into V_H and every directed edge $u \to v$ of G to a directed edge $h(u) \to h(v)$ of H.

able. The two main results regarding this question are collected in the following theorem[34].

Theorem 1.29 Let L be a set of finite, simple, undirected graphs.

(1) If L has a decidable MS_2-satisfiability problem, then it has bounded tree-width.

(2) If L has a decidable C_2MS-satisfiability problem, then it has bounded clique-width. □

Assertion (1) is proved by Seese in [See91]. He asks in this article whether every set L having a decidable MS-satisfiability problem (which is weaker than having a decidable C_2MS-satisfiability problem) is "tree-like", which is actually equivalent (by the Equationality Theorem for the VR algebra presented below in Section 1.7.3) to having bounded clique-width. Assertion (2) proved in [CouOum] is thus a partial answer to Seese's question.

1.7 Graph transductions

The theory of formal languages studies finite descriptions of languages (sets of words or terms) by grammars and automata, and also finite descriptions of transformations of these objects. Motivations come from the theories of coding, of compilation, of computational linguistics, just to cite a few. The finite devices that specify these transformations are called *transducers* and the corresponding transformations of words and terms are called *transductions*. Typical questions are the following:

Is a given class of transductions closed under composition? under inverse?

Does it preserve a given family of languages?

Is this family the set of images of a particular language (called a generator*) under the transductions of the class?*

Is the equality of the transductions defined by two transducers of a certain type decidable?

In the study of sets of words, *rational transductions* play a prominent role originating from the work by Nivat [Niv]. The inverse of a rational transduction and the composition of two rational transductions are rational transductions. The families of regular and of context-free languages are preserved under rational transductions, and there exist context-free languages (like the one defined by the equation $L = f(L, L) \cup a$ where $L \subseteq A^*$ and A consists of a, f, parentheses and comma), whose images under all rational transductions are all context-free languages. Transductions of words are studied in the books [*Ber] and [*Sak]. Transductions of terms, usually called *tree transductions*, are studied in [*Com+] and [*GecSte].

[34] We recall (from Section 1.3.1) that the acronym C_2MS refers to MS logic extended by the even cardinality set predicate.

Transducers are usually based on finite automata or on more complicated devices like macros (see, e.g, the *macro tree transducers* in [EngMan99]). Can one define similar notions for graphs? Since graphs can be denoted by terms, one can use transductions of terms to specify transductions of graphs (as, e.g., in [*Eng94, DreEng]). However, doing this requires for processing the input a *parsing* step that is algorithmically difficult (cf. Sections 1.5 and 6.2). For building the output, each term produced by such a transduction must be evaluated into a graph. The main drawback of the detour through terms over (necessarily) finite signatures is that it limits the input and output graphs of transductions to have bounded tree-width or clique-width. It is natural to try to avoid such a detour and to define transducers that work "directly" on graphs, by traversing them according to some rules and by starting from some specified vertex. However no notion of finite graph automaton has been defined that would generalize conveniently finite automata on words and terms. Monadic second-order logic offers a powerful alternative. In this section, we define *monadic second-order transductions* by means of examples more than formally. These transductions have good interactions with the HR- and VR-equational sets (our "context-free sets of graphs") that follow from the *Equationality Theorem* presented below in Section 1.7.3. It will be shown in Chapter 8 that every (functional) monadic second-order transduction of graphs of bounded tree-width or clique-width can be realized by a macro tree transducer on the level of terms.

1.7.1 Examples of monadic second-order transductions

Monadic second-order transductions are transformations of graphs specified by monadic second-order formulas. The basic notion is that of a monadic second-order transduction of relational structures over a fixed set of relation symbols. It applies to graphs faithfully represented by relational structures.

All examples and results presented in this section (Section 1.7) will concern simple graphs, for which $G \mapsto \lfloor G \rfloor$ is a faithful representation. Hence, we will consider mappings f from simple graphs to simple graphs such that, for every G, the structure $\lfloor f(G) \rfloor$ representing its image under f is defined from $\lfloor G \rfloor$ by monadic second-order formulas.

The simplest case is when:

$$\begin{array}{lcl} \lfloor f(G) \rfloor & := & \langle V_{f(G)}, edg_{f(G)} \rangle, \\ V_{f(G)} & := & \{u \in V_G \mid \lfloor G \rfloor \models \delta(u)\}, \\ edg_{f(G)} & := & \{(u,v) \in V_G \times V_G \mid \lfloor G \rfloor \models \delta(u) \wedge \delta(v) \wedge \theta(u,v)\}, \end{array}$$

where δ and θ are monadic second-order formulas with free variables u, and u and v respectively. The formula δ defines the set of vertices of $f(G)$ as a subset of V_G and the formula θ defines the edge relation of $f(G)$ in terms of that of G. The mapping f is thus specified by a pair $\langle \delta, \theta \rangle$ of monadic second-order formulas. We will say that f is a *monadic second-order transduction* and that $\langle \delta, \theta \rangle$ is its *definition scheme*. If δ and θ are first-order formulas, we will

say that f is a *first-order transduction*. The general definition is actually more complicated. We introduce it step by step by giving several examples.

Example 1.30 (Edge-complement)
The *edge-complement* associates with a simple, undirected and loop-free graph G, the simple, undirected and loop-free graph $\overline{G}$ such that $V_{\overline{G}} = V_G$ and $u - v$ is an edge of $\overline{G}$ if and only if $u \neq v$ and $u - v$ is not an edge of G. This transformation is a first-order transduction with definition scheme $\langle \delta, \theta \rangle$ where:

$$\begin{aligned} &\delta(x) \text{ is the Boolean constant } \mathit{True}, \\ &\theta(x,y) \text{ is the formula } x \neq y \wedge \neg \mathit{edg}(x,y). \end{aligned}$$

Example 1.31 (Elimination of loops and isolated vertices)
The transformation that eliminates loops and then, isolated vertices is a first-order transduction with definition scheme $\langle \delta, \theta \rangle$ where:

$$\begin{aligned} &\delta(x) \text{ is the formula } \exists y \Big(\big(\mathit{edg}(x,y) \vee \mathit{edg}(y,x) \big) \wedge x \neq y \Big), \\ &\theta(x,y) \text{ is the formula } \mathit{edg}(x,y) \wedge x \neq y. \end{aligned}$$

Example 1.32 (Transitive closure of a directed graph)
For every directed graph G, we let G^+ be its transitive closure, the simple graph defined by:

$$\begin{aligned} V_{G^+} &:= V_G, \\ \mathit{edg}_{G^+} &:= \mathit{edg}_G^+ \\ &:= \{(u,v) \mid \text{ there is in } G \text{ a nonempty directed path}^{35} \text{ from } u \text{ to } v\}. \end{aligned}$$

Note that u has a loop in G^+ if it belongs to a directed cycle in G. The mapping $G \mapsto G^+$ is defined by the definition scheme $\langle \delta, \theta \rangle$ where:

$$\begin{aligned} &\delta(x) \text{ is the formula } \mathit{True}, \\ &\theta(x,y) \text{ is the formula}^{36} \ \mathit{edg}(x,y) \vee \exists z (\mathit{edg}(x,z) \wedge \mathrm{TC}[\mathit{edg}; z, y]). \end{aligned}$$

We use here the definition of the reflexive and transitive closure of a binary relation by a monadic second-order formula presented in Section 1.3.1.

Example 1.33 (Transitive reduction of a directed acyclic graph)
A *directed acyclic graph* (a *DAG*) is a simple directed graph without directed cycles. Every such finite graph G has a unique minimal subgraph H such that $H^+ = G^+$. It is called the *transitive reduction* of G and is denoted by $\mathit{Red}(G)$. (The Hasse diagram of a partial order $\langle D, \leq \rangle$ is a graphical representation of

[35] i.e., a sequence of pairwise distinct vertices $w_1, w_2, \ldots, w_n$ such that $u = w_1$, $v = w_n$ and $w_i \to w_{i+1}$ in G for each $i = 1, \ldots, n-1$. A directed cycle is a sequence of this form with $w_1 = w_n$, $n \geq 2$, $w_i \neq w_j$ for $1 \leq i \leq j \leq n-1$.

[36] The atomic formula $\mathit{edg}(x,y)$ in $\theta(x,y)$ can be omitted, but putting it in makes the formula more clear.

the transitive reduction of the directed acyclic graph $\langle D, < \rangle$.) The edge relation of $Red(G)$ is characterized by:

$$edg_{Red(G)}(x, y) \iff edg(x, y) \wedge \neg\exists z(edg_{G^+}(x, z) \wedge edg_{G^+}(z, y)),$$

hence is defined in G by a monadic second-order formula θ' built with the formula θ of the previous example. It follows that the mapping *Red* from directed graphs to directed graphs is a partial function that is a monadic second-order transduction specified by a sentence χ and two formulas δ, θ' such that:

$\lfloor G \rfloor \models \chi$	if and only if it is acyclic (hence if and only if $Red(G)$ is well defined),
$\lfloor G \rfloor \models \delta(u)$	for every u in V_G (so that $V_{Red(G)} = V_G$),
$\lfloor G \rfloor \models \theta'(u, v)$	if and only if $Red(G)$ has an edge $u \to v$.

□

It is frequently necessary to consider functions f that assign a graph to the pair of a graph G and a set of vertices X. In this case, the definition scheme consists of formulas χ, δ and θ with a free set variable X called a *parameter*. The formula χ expresses the conditions to be verified by G and X so that $f(G, X)$ be defined. Here is an example.

Example 1.34 (The largest connected subgraph of G containing X)
The partial function f such that $f(G, X)$ is the largest connected induced subgraph of G containing a nonempty subset X of V_G is a monadic second-order transduction with parameter X. Its definition scheme is $\langle \chi, \delta, \theta \rangle$ where:

$\chi(X)$ is the formula $(\exists x.\, x \in X) \wedge \exists Y\big(X \subseteq Y \wedge \mathrm{CONN}(Y)\big)$,[37]

$\delta(X, x)$ is the formula $\exists Y\big(X \subseteq Y \wedge x \in Y \wedge \mathrm{CONN}(Y)\big)$,

$\theta(X, x, y)$ is $\exists Y\big(X \subseteq Y \wedge x \in Y \wedge y \in Y \wedge edg(x, y) \wedge \mathrm{CONN}(Y)\big)$. □

In the previous example, a parameter X is necessary because the function f to be defined does not depend only on G but also on a set of vertices. However, in some cases, parameters may be necessary even if the output graphs depend only, up to isomorphism, on the input graphs (and not on the values of the parameters). Here is an example of such a case.

Example 1.35 (The DAG of strongly connected components of a directed graph)
Let G be a directed graph. Let $\approx$ be the equivalence relation on V_G defined by:

$u \approx v$ if and only if $u = v$ or there exists a directed path from u to v and a directed path from v to u.

[37] The formula $\mathrm{CONN}(Y)$ is defined in Section 1.3.1. We use $X \subseteq Y$ as a shorthand for $\forall x(x \in X \Rightarrow x \in Y)$.

An induced subgraph $G[X]$ where X is an equivalence class of this relation is called a *strongly connected component*. The DAG of strongly connected components of G is the *quotient graph* $G/\approx$ defined as follows: its vertices are the strongly connected components; there is in $G/\approx$ an edge $X \to Y$ if and only if $X \neq Y$ and $u \to v$ in G for some $u \in X$ and $v \in Y$. Our objective is to prove that there exists a monadic second-order transduction associating with every graph G a graph H isomorphic to $G/\approx$. Its definition uses a parameter X for denoting sets $U \subseteq V_G$ required to contain one and only one vertex of each strongly connected component. Such sets are used as sets of vertices of H and its edges are defined accordingly. We use the following formulas:

$Eq(x, y)$ is the auxiliary formula $\mathrm{TC}[edg; x, y] \wedge \mathrm{TC}[edg; y, x]$,

$\chi(X)$ is defined as:

$$\forall x, y\Big(\big(x \in X \wedge y \in X \wedge Eq(x, y)\big) \Rightarrow x = y\Big) \wedge \forall x \exists y\Big(Eq(x, y) \wedge y \in X\Big),$$

$\delta(X, x)$ is defined as: $x \in X$,

$\theta(X, x, y)$ is defined as:

$$x \in X \wedge y \in X \wedge \neg Eq(x, y) \wedge \exists z, z'\Big(edg(z, z') \wedge Eq(x, z) \wedge Eq(y, z')\Big).$$

The definition scheme $\langle \chi, \delta, \theta \rangle$ specifies a monadic second-order transduction that associates with a directed graph G and a "well-chosen" subset U of V_G a graph $f(G, U)$ with vertex set U that is isomorphic to $G/\approx$ ($u \in U$ corresponds to an equivalence class, hence to a vertex of $G/\approx$). It is clear that for every graph G there exists a set U satisfying χ, and that, for any two sets U and U' satisfying χ, the graphs $f(G, U)$ and $f(G, U')$ are isomorphic (the corresponding bijection $h : U \to U'$ is defined by $h(u) = v$ if and only if $u \in U$, $v \in U'$ and $u \approx v$).

This construction extends actually to every equivalence relation definable by a monadic second-order formula in place of $\approx$ and proves that the mapping that associates with a graph G its quotient $G/\approx$ by a monadic second-order definable equivalence relation $\approx$ is a monadic second-order transduction. □

In all the above examples, the set of vertices of the output graph is a subset of the set of vertices of the input graph. The general definition of a monadic second-order transduction includes the possibility of *enlarging the input graph*, by "copying it" a fixed number of times.

A *k-copying monadic second-order transduction* associates with a graph G a graph H such that:

$$\begin{aligned} V_H &:= (V_1 \times \{1\}) \cup \cdots \cup (V_k \times \{k\}) \\ edg_H &:= \{((u, i), (v, j)) \mid 1 \leq i, j \leq k,\ (u, v) \in E_{i,j}\} \end{aligned}$$

where the sets $V_1, \ldots, V_k \subseteq V_G$ and the relations $E_{i,j} \subseteq V_G \times V_G$ are defined in G by monadic second-order formulas, respectively $\delta_1, \ldots, \delta_k$ and $\theta_{i,j}$ for $1 \leq i, j \leq k$ and possibly written with parameters. Let us give an example.

Example 1.36 (Graph duplication)
For every simple directed graph G, its *duplication* is the simple graph $H = dup(G)$ defined as follows:

$V_H := V_G \times \{1, 2\}$,
the edges of H are the edges of each copy of G, together with the edges $(u, 1) \to (u, 2)$ for all $u \in V_G$.

The mapping *dup* has the definition scheme $\langle \delta_1, \delta_2, \theta_{1,1}, \theta_{1,2}, \theta_{2,1}, \theta_{2,2} \rangle$ where:

$\delta_1(x)$ and $\delta_2(x)$ are both the Boolean constant *True*,
$\theta_{1,1}(x, y)$ and $\theta_{2,2}(x, y)$ are both $edg(x, y)$,
$\theta_{1,2}(x, y)$ is $x = y$,
$\theta_{2,1}(x, y)$ is the Boolean constant *False*.

The formulas δ_1 and δ_2 define V_1 and V_2 as V_G; the formulas $\theta_{i,j}$ define $E_{1,1} := E_{2,2} := edg_G$, $E_{1,2} := \{(u, u) \mid u \in V_G\}$, and $E_{2,1} := \emptyset$. Informally H consists of two disjoint copies of G (i.e, $G \oplus G$) together with an edge from any vertex of the first copy to the corresponding vertex of the second copy. □

The general definition of a monadic second-order transduction combines the above presented features. To summarize, a monadic second-order transduction associates with a relational structure S and subsets $U_1, \ldots, U_m$ of its domain D_S that must satisfy a monadic second-order formula $\chi(X_1, \ldots, X_m)$, a relational structure $T = f(S, U_1, \ldots, U_m)$ defined as follows. Its domain is $D_T := (D_1 \times \{1\}) \cup \cdots \cup (D_k \times \{k\})$ where each set D_i is defined as $\{d \in D_S \mid S \models \delta_i(U_1, \ldots, U_m, d)\}$ for some monadic second-order formula δ_i. If R is an n-ary symbol of the relational signature of T, the corresponding n-ary relation on D_T is defined as

$$\bigcup_{i_1,\ldots,i_n \in [k]} \{((d_1, i_1), \ldots, (d_n, i_n)) \mid d_1 \in D_{i_1}, \ldots, d_n \in D_{i_n},$$

$$S \models \theta_{R,i_1,\ldots,i_n}(U_1, \ldots, U_m, d_1, \ldots, d_n)\},$$

where each $\theta_{R,i_1,\ldots,i_n}$ is a monadic second-order formula.

Such a transduction f is specified by a tuple of formulas called a *definition scheme* of the form $\langle \chi, \delta_1, \ldots, \delta_k, (\theta_w)_{w \in W} \rangle$ where W consists of all tuples $(R, i_1, \ldots, i_{\rho(R)})$ such that R is a relation symbol of T and $i_1, \ldots, i_{\rho(R)} \in [k]$. The following fact is clear from the definitions:

Fact 1.37 For every monadic second-order transduction f there exists an integer k such that, if f transforms a relational structure S into a relational structure T, then $|D_T| \leq k \cdot |D_S|$. □

We have $k = 1$ in the first six examples and $k = 2$ in the last one. We now give another example, with $k = 1$ and without parameters.

Example 1.38 (The cograph denoted by a term)

As a final example we consider the mapping that evaluates a term t in $T(\{\oplus, \otimes, \mathbf{1}\})$ into the cograph $val(t)$ (cf. Section 1.1.2 and Example 1.7). First of all we must explain how t is represented by a relational structure. We let $\lfloor t \rfloor = \langle N_t, son_t, lab_{\oplus t}, lab_{\otimes t}, lab_{\mathbf{1}t} \rangle$ be the relational structure such that:

- N_t is $Pos(t)$, the set of positions of t, i.e., of occurrences of symbols from $\{\oplus, \otimes, \mathbf{1}\}$; we will consider N_t as the set of nodes of a rooted labelled tree representing t and also denoted by t;
- son_t is the binary relation such that $son_t(u, v)$ holds if and only if v is a son of u in the tree[38] t;
- $lab_{\oplus t}$, $lab_{\otimes t}$ and $lab_{\mathbf{1}t}$ are the unary relations such that $lab_{\oplus t}(u)$ holds if and only if u is an occurrence of $\oplus$ in t (i.e., is labelled by $\oplus$ as a node of the tree t) and similarly for $lab_{\otimes t}$ and $lab_{\mathbf{1}t}$.

Since the operations $\oplus$ and $\otimes$ are commutative, we need not express, when v is a son of u, whether it is the left or the right son. Hence we can use a simpler representation than the general one to be defined in Section 5.1.1. Here is an example. We let:

$$s \quad := \quad \Big(\big(\mathbf{1}_1 \otimes_2 \mathbf{1}_3 \big) \oplus_4 \mathbf{1}_5 \Big) \otimes_6 \Big(\mathbf{1}_7 \oplus_8 \mathbf{1}_9 \Big)$$

where we number from left to right the nine occurrences of $\mathbf{1}$, $\oplus$, $\otimes$ (this numbering is indicated by subscripts). The corresponding labelled tree is shown in Figure 1.5. Then

$$\begin{array}{lcl} \lfloor s \rfloor & = & \langle [9], son, lab_{\oplus}, lab_{\otimes}, lab_{\mathbf{1}} \rangle \text{ with} \\ son & = & \{(6,4),(6,8),(4,2),(4,5),(2,1),(2,3),(8,7),(8,9)\}, \\ lab_{\oplus} & = & \{4,8\}, \\ lab_{\otimes} & = & \{2,6\}, \\ lab_{\mathbf{1}} & = & \{1,3,5,7,9\}. \end{array}$$

In the general case, the cograph $G = val(t)$ that is the value of a term t in $T(\{\oplus, \otimes, \mathbf{1}\})$ can be defined from $\lfloor t \rfloor$ as follows:

> V_G is the set of elements of N_t that are labelled by $\mathbf{1}$ (i.e, that are the leaves of the tree t); for distinct vertices u and v of G, there is an edge between u and v if and only if the least common ancestor of u and v in t is labelled by $\otimes$.

In the above example of s, the vertices of the cograph $val(s)$ are $1, 3, 5, 7, 9$. There is an edge between 3 and 9 because their least common ancestor in s is 6 which is labelled by $\otimes$. There is no edge between 1 and 5 because their least common ancestor is 4, labelled by $\oplus$.

[38] We describe the relations between occurrences of symbols in t with the terminology of trees. Chapter 2 will detail the terminology about terms and the trees that represent them.

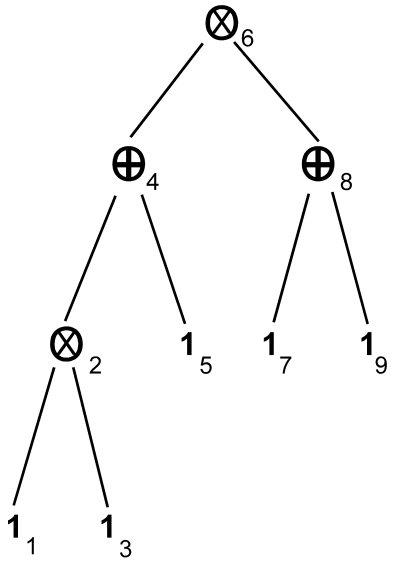

Figure 1.5: The term s as a labelled tree.

The monadic second-order transduction that associates with $\lfloor t \rfloor$, for any term t in $T(\{\oplus, \otimes, \mathbf{1}\})$, the relational structure $\lfloor val(t) \rfloor$ is specified by the definition scheme $\langle True, \delta, \theta \rangle$ where:

$$\delta(x) \text{ is the formula } \; lab_{\mathbf{1}}(x),$$
$$\theta(x,y) \text{ is the formula } \; x \neq y \wedge \exists z(\mathrm{LCA}(x,y,z) \wedge lab_{\otimes}(z)).$$

In this writing, $\mathrm{LCA}(x,y,z)$ stands for the following formula expressing that z is the least common ancestor of x and y:

$$\mathrm{TC}[son; z, x] \wedge \mathrm{TC}[son; z, y] \wedge$$
$$\forall w\big((\mathrm{TC}[son; w, x] \wedge \mathrm{TC}[son; w, y]) \Rightarrow \mathrm{TC}[son; w, z]\big).$$

This formula is a straightforward translation of the definition of the least common ancestor. Note that if x is an ancestor of y or if $x = y$, then $\mathrm{LCA}(x,y,x)$ is valid. Furthermore $\mathrm{LCA}(x,y,z)$ defines z in a unique way from x and y.

1.7.2 The main properties of monadic second-order transductions

A *monadic second-order transduction* is a subset f of $\mathcal{S} \times \mathcal{T}$ where $\mathcal{S}$ and $\mathcal{T}$ are classes of graphs or, more generally, of relational structures, that is specified as explained in the examples by monadic second-order formulas. If $L \subseteq \mathcal{S}$ and $f \subseteq \mathcal{S} \times \mathcal{T}$, we let $f(L) := \{T \mid (S,T) \in f,\ S \in L\}$ be the *image* of L under f. Hence, f can also be seen as the multivalued mapping $\widehat{f} : \mathcal{S} \to \mathcal{P}(\mathcal{T})$

such that $\widehat{f}(G) := f(\{G\})$. In Examples 1.30, 1.31 and 1.32, $\widehat{f}(G)$ consists of a unique graph. In Example 1.33 it consists of at most one graph and in Example 1.34 of several graphs (the connected components). However, a monadic second-order transduction is always based on a (single-valued) partial function that transforms *parametrized relational structures* into relational structures. A parametrized relational structure is a tuple $(S, A_1, \ldots, A_n)$ consisting of a relational structure S and an n-tuple of subsets of its domain D_S. From a partial function $\overline{f}$ that transforms $(S, A_1, \ldots, A_n)$ into $\overline{f}(S, A_1, \ldots, A_n)$ in $\mathcal{T}$ for S in $\mathcal{S}$ and for certain subsets $A_1, \ldots, A_n$ of D_S, we define f as the transduction $\{(S, \overline{f}(S, A_1, \ldots, A_n)) \mid A_1, \ldots, A_n \subseteq D_S \text{ and } \overline{f}(S, A_1, \ldots, A_n) \text{ is defined}\}$. We also say that $\overline{f}$ and $\widehat{f}$ are monadic second-order transductions.

The *composition* of two monadic second-order transductions is the composition of the corresponding binary relations: if $f \subseteq \mathcal{S} \times \mathcal{T}$ and $g \subseteq \mathcal{T} \times \mathcal{U}$, then $f \cdot g := \{(S, U) \mid (S, T) \in f \text{ and } (T, U) \in g \text{ for some } T \in \mathcal{T}\}$. If f and g are functional, we will also use the notation $g \circ f$ for $f \cdot g$.

Theorem 1.39 The composition of two monadic second-order transductions is a monadic second-order transduction. □

A more precise statement showing how parameters are handled in the composition of monadic second-order transductions will be given in Chapter 7 (Theorem 7.14).

Theorem 1.40 (Backwards Translation Theorem)
If the set $L \subseteq \mathcal{T}$ is MS-definable and $f \subseteq \mathcal{S} \times \mathcal{T}$ is a monadic second-order transduction, then the set $f^{-1}(L) := \{S \in \mathcal{S} \mid \widehat{f}(S) \cap L \neq \emptyset\}$ is MS-definable. □

Theorem 1.40 is formally a consequence of Theorem 1.39 but it is actually used for proving Theorem 1.39 (see Chapter 7). We call Theorem 1.40 the Backwards Translation Theorem because its proof yields an algorithm that transforms a formula defining L into one defining $f^{-1}(L)$. We now give the idea of its proof. Let us go back to the edge-complement transformation (Example 1.30). It is a first-order transduction that transforms G into $\overline{G}$. If P is a first-order graph property, then the property Q defined by $Q(G)$ if and only if $P(\overline{G})$ is also first-order. The edge relation of $\overline{G}$ is defined by $edg_{\overline{G}}(u, v) :\Longleftrightarrow u \neq v \wedge \neg edg_G(u, v)$. Hence one obtains a first-order sentence expressing Q by replacing in the given first-order sentence expressing P each atomic formula of the form $edg(x, y)$ by $(x \neq y \wedge \neg edg(x, y))$. Here is an example. We let P be defined by the sentence:

$$\exists x, y, z \Big(x \neq y \wedge y \neq z \wedge x \neq z \wedge \\ edg(x, y) \wedge edg(y, z) \wedge \neg edg(x, z) \Big)$$

expressing that an undirected graph G has an induced path P_3 (of the form

$\bullet - \bullet - \bullet$). The sentence defining Q is then:

$$\exists x, y, z \Big(x \neq y \wedge y \neq z \wedge x \neq z \wedge \big(x \neq y \wedge \neg edg(x,y)\big) \wedge \big(y \neq z \wedge \neg edg(y,z)\big) \wedge \neg\big(x \neq z \wedge \neg edg(x,z)\big)\Big).$$

It can actually be simplified into:

$$\exists x, y, z \Big(x \neq y \wedge y \neq z \wedge x \neq z \wedge \neg edg(x,y) \wedge \neg edg(y,z) \wedge edg(x,z)\Big)$$

expressing that G has an induced subgraph of the form $\bullet\!-\!\bullet\ \bullet$. This construction extends easily to monadic second-order transductions without parameters, like those of Examples 1.31, 1.32 and 1.33.

Let us now assume that f is defined by a definition scheme of the form $\langle \chi(X), \mathit{True}, \theta(X,x,y)\rangle$ with one parameter X. Here $\chi(X)$ is a formula that imposes some conditions on the parameter X. In Example 1.34, the condition "X denotes a singleton" might be imposed on X: the corresponding transduction associates with each vertex its connected component.

Let β be a sentence and let L be the set of graphs such that $\lfloor G \rfloor \models \beta$. Let then $\beta^{\#}$ be the formula with free variable X obtained by replacing in β every atomic formula $edg(x,y)$ by $\theta(X,x,y)$ (by using appropriate substitutions, and, if necessary, renamings of bound variables in $\theta(X,x,y)$). For $A \subseteq V_G$, the graph $f(G,A)$ is well defined if and only if $\lfloor G \rfloor \models \chi(A)$, and then, $f(G,A) \models \beta$ if and only if $\lfloor G \rfloor \models \beta^{\#}(A)$. It follows that $f^{-1}(L)$ is defined by the sentence $\exists X \big(\chi(X) \wedge \beta^{\#}(X)\big)$.

1.7.3 The Equationality Theorem

The Equationality Theorems for the VR and the HR algebras (and for an algebra of relational structures that generalizes the VR algebra) are among the main results established in this book.

As an introduction to the Equationality Theorem for the VR algebra, we recall that the mapping that associates the cograph $val(t)$ with a term t in $T(\{\oplus, \otimes, \mathbf{1}\})$ is a monadic second-order transduction (Example 1.38). More generally:

Theorem 1.41 For every k, the mapping that associates with a term t in $T(F^{\mathrm{VR}}_{[k]})$ the p-graph $t_{\mathbb{GP}}$ is a monadic second-order transduction. □

This implies that the set of graphs of clique-width at most k, which is a VR-equational set, is the image of the set of terms (over some finite signature) under a monadic second-order transduction. The following result includes a converse of this result and is very important for the theories of graph structuring and of graph grammars.

Theorem 1.42 (Equationality Theorem for the VR algebra)
A set of simple graphs is VR-equational if and only if it is the image of the set of binary rooted trees under a monadic second-order transduction. □

As an immediate consequence and by using Theorem 1.39, we obtain that the image of a VR-equational set under a monadic second-order transduction is again VR-equational. Note that this implies, as a special case, the logical version of the Filtering Theorem (Theorem 1.22): for a sentence χ, the transduction $f_\chi = \{(G, G) \mid \lfloor G \rfloor \models \chi\}$ is a monadic second-order transduction with definition scheme $\langle \chi, \mathit{True}, \mathit{edg}(x, y) \rangle$, and $f_\chi(L) = \{G \in L \mid \lfloor G \rfloor \models \chi\}$ for every set of graphs L. Another immediate consequence, using Proposition 1.18, is the following.

Corollary 1.43 A set of simple graphs has bounded clique-width if and only if it is included in the image of the set of binary rooted trees under a monadic second-order transduction. □

Thus, again using Theorem 1.39, the image of a set of graphs of bounded clique-width under a monadic second-order transduction from graphs to graphs has bounded clique-width.

In Theorem 1.42 and Corollary 1.43, one can replace "the set of binary rooted trees"[39] by "the set of trees" or by "$T(F)$ where F is any finite signature with at least one constant symbol and at least one symbol of arity at least 2".

1.8 Monadic second-order logic with edge set quantifications

If a graph G is represented by a relational structure whose domain also contains the edges, instead of by $\lfloor G \rfloor$, then the expressive power of monadic second-order logic is increased, even for expressing properties of simple graphs. We will also compare the four types of monadic second-order transductions obtained by representing input and output graphs G either by $\lfloor G \rfloor$ or by the alternative structure denoted by $\lceil G \rceil$.

1.8.1 Expressing graph properties with edge set quantifications

For every undirected graph G, we let $\lceil G \rceil$ be the pair $\langle V_G \cup E_G, in_G \rangle$ where[40] $in_G = \{(e, u) \mid e \in E_G, u \in V_G,\ u \text{ is an end vertex of } e\}$. If G has several edges (called *multiple edges*) between two vertices, these edges are distinct elements of the domain of $\lceil G \rceil$. The structure $\lceil G \rceil$ can be seen as $\lfloor \mathit{Inc}(G) \rfloor$ where $\mathit{Inc}(G)$

[39] A rooted tree is directed in such a way that the root is the unique node of indegree 0 and every node is accessible from the root by a directed path. It is *binary* if each node has outdegree 0 or 2. The associated (undirected) tree has degree at most 3.

[40] Recall that V_G is the set of vertices and E_G is the set of edges.

is a bipartite directed graph called the *incidence graph* of G, whose edge relation is denoted by the binary relation symbol in. In a relational structure $S = \langle D_S, in_S \rangle$ isomorphic to $\lceil G \rceil$ for some graph G, the elements of D_S corresponding to edges are those, say u, such that $(u, v) \in in_S$ for some v. The element u corresponds to a loop if and only if there is a single such v. It follows that G can be reconstructed from S in a unique way. Thus, the representation of G by $\lceil G \rceil$ is faithful for all undirected graphs.

For directed graphs, we will use $\lceil G \rceil := \langle V_G \cup E_G, in_{1G}, in_{2G} \rangle$ where $(e, u) \in in_{1G}$ (resp. $(e, u) \in in_{2G}$) if and only if u is the tail[41] (resp. the head) of e. Hence $\lceil G \rceil$ can be seen as a directed bipartite graph, also denoted by $Inc(G)$, with edges labelled either by 1 or by 2. Loops in G are multiple edges (with different labels) in $Inc(G)$.

In this setting, edges are considered, like vertices, as objects that form a graph and not as the pairs of some binary relation over vertices. In particular, we do not consider an undirected edge as a pair of opposite directed edges.

Graph properties can be expressed logically, either via the representation of a graph G by $\lceil G \rceil$, or via the initially defined representation $\lfloor G \rfloor := \langle V_G, edg_G \rangle$. The representation $\lfloor G \rfloor$ only allows quantification on vertices and on sets of vertices in monadic second-order formulas, whereas the representation $\lceil G \rceil$ also allows quantification on edges and sets of edges. A graph property is MS_2-*expressible* if it is expressible by a monadic second-order formula interpreted in $\lceil G \rceil$. The index 2 refers to the possibility of "two types of quantification", on sets of vertices and sets of edges. A property is MS_1-*expressible* if it is by a monadic second-order formula interpreted in $\lfloor G \rfloor$. Unless for emphasizing the contrast with MS_2, we will write MS instead of MS_1.[42]

Let us stress that we *do not modify* the logical language, but only the representation of graphs by relational structures. Since an incidence graph is a graph, we still deal with a single language that we use to express formally graph properties. We now compare the power of these two ways of expressing graph properties. It is clear that a property of a graph G like "for every two vertices u, v, there are no more than 3 edges from u to v" cannot be expressed by any sentence interpreted in $\lfloor G \rfloor$ because this relational structure cannot identify the existence of multiple edges, and thus no sentence can take into account their multiplicity. However, even for expressing properties of simple graphs, monadic second-order sentences with edge set quantifications are more powerful. The property that a loop-free undirected graph G has a perfect matching is equivalent to $\lceil G \rceil \models \exists X.\, \psi$ where ψ is a formula with free variable X expressing that X is a set of edges and that for every vertex u there exists a unique $e \in X$ such that $(e, u) \in in_G$. The formula ψ is easy to write with first-order quantifications only. However, there is no monadic second-order sentence φ expressing this property by $\lfloor G \rfloor \models \varphi$. The formal proof of this assertion (to be done in

[41] If e is an edge directed from u to v, then we say that u is its *tail* and that v is its *head*.

[42] By an MS_2 *formula*, we mean a monadic second-order formula written with the binary relation symbols in, in_1, in_2, intended to be interpreted in logical structures of the form $\lceil G \rceil$. An MS_1 *formula* is written with the binary relation symbol edg and is to be interpreted in $\lfloor G \rfloor$.

Chapter 5) is based on the observation that the complete bipartite graph $K_{n,m}$ has a perfect matching if and only if $n = m$, and on the theorem saying that no monadic second-order formula can express that two sets have the same cardinality. It is easy to express that a *given* set of edges, say X, is a perfect matching, and this is what formula ψ does. But one cannot replace "there exists a set of edges satisfying ψ" by an MS formula without edge set quantification. The property "G has a Hamiltonian cycle" can be expressed similarly by an MS formula interpreted in $\lceil G \rceil$. The graphs $K_{n,m}$ can be used as counter-examples, as above for perfect matchings, to prove that this is not possible by an MS formula over $\lfloor G \rfloor$.

It is clear that every MS_1-expressible graph property is MS_2-expressible. Although some properties of simple graphs are MS_2-expressible but not MS_1-expressible, MS_2 formulas are in many cases no more expressive than MS_1 formulas.

Theorem 1.44 (Sparseness Theorem)
Let L be a set of simple graphs that are all, either planar, or of degree at most k, or of tree-width at most k for some fixed k. Every MS_2 sentence φ can be translated into an MS_1 sentence ψ such that for every graph G in L:

$$\lceil G \rceil \models \varphi \text{ if and only if } \lfloor G \rfloor \models \psi.$$

□

This result actually extends to sets of *uniformly sparse* graphs, as we will prove in Section 9.4.

1.8.2 Monadic second-order transductions over incidence graphs

For expressing properties of graphs G, we can choose between the two representing relational structures $\lfloor G \rfloor$ and $\lceil G \rceil$. For defining monadic second-order graph transductions, we get thus four possibilities arising from two possible representations for the input as well as for the output. All examples of Section 1.7.1 use the first representation for the input and the output. We give below examples using $\lceil G \rceil$. We first fix some notation.

A graph transduction f is an *$MS_{i,j}$-transduction* where $i, j \in \{1, 2\}$, if there exists a monadic second-order transduction g such that $(G, H) \in f$ if and only if $(S, T) \in g$ where S is $\lfloor G \rfloor$ if $i = 1$ and $\lceil G \rceil$ if $i = 2$, and similarly, T is $\lfloor H \rfloor$ if $j = 1$ and $\lceil H \rceil$ if $j = 2$. Hence the indices i and j indicate which representations are used, respectively for the input and the output graphs.

It is clear that every $\mathrm{MS}_{1,j}$-transduction is also an $\mathrm{MS}_{2,j}$-transduction, because every MS_1 formula can be rewritten into an equivalent MS_2 formula. Hence changing in this way 1 into 2 makes "easier" the task of writing formulas to specify a transduction. For the "output" side we get that every $\mathrm{MS}_{i,2}$-transduction is an $\mathrm{MS}_{i,1}$-transduction, and not vice-versa as one might think, because in the former case the transduction must define the edges (and not only

the vertices) from elements of the input structure, either $\lfloor G \rfloor$ or $\lceil G \rceil$. We have the following inclusions of classes of monadic second-order transductions:

$$\mathrm{MS}_{1,2} \subseteq \mathrm{MS}_{1,1} \subseteq \mathrm{MS}_{2,1}$$
$$\mathrm{MS}_{1,2} \subseteq \mathrm{MS}_{2,2} \subseteq \mathrm{MS}_{2,1}.$$

We will give examples proving that these inclusions are proper, that $\mathrm{MS}_{1,1}$ and $\mathrm{MS}_{2,2}$ are incomparable, that $\mathrm{MS}_{1,1} \cup \mathrm{MS}_{2,2}$ is a proper subclass of $\mathrm{MS}_{2,1}$ and that $\mathrm{MS}_{1,2}$ is a proper subclass of $\mathrm{MS}_{1,1} \cap \mathrm{MS}_{2,2}$.

Example 1.45 (The line graph transduction)

The *line graph* $Line(G)$ of an undirected graph G is the loop-free undirected graph H such that $V_H = E_G$ and e, f are adjacent vertices of H if and only if they have at least one common vertex as edges of G. The mapping $\lceil G \rceil \to \lfloor H \rfloor$ is an $\mathrm{MS}_{2,1}$-transduction with definition scheme $\langle \chi, \delta, \theta_{edg} \rangle$ such that:

$$\begin{array}{lll} \chi & :\Longleftrightarrow & \mathit{True}, \\ \delta(x) & :\Longleftrightarrow & \exists z.\, in(x,z), \\ \theta_{edg}(x,y) & :\Longleftrightarrow & x \neq y \wedge \exists z(in(x,z) \wedge in(y,z)). \end{array}$$

Could one use $\lfloor G \rfloor$ instead of $\lceil G \rceil$ for the input? The answer is no by Fact 1.37 because for arbitrary graphs G, if $H = Line(G)$ then we do not have $|D_{\lfloor H \rfloor}| = O(|D_{\lfloor G \rfloor}|)$ since $D_{\lfloor H \rfloor} = V_H = E_G$ and $D_{\lfloor G \rfloor} = V_G$.

Could one use $\lceil H \rceil$ instead of $\lfloor H \rfloor$ for the output? Again the answer is no by a similar argument: if $G = K_{1,n}$, then $|D_{\lceil G \rceil}| = 2n + 1$, $|V_H| = n$, $|E_H| = n(n-1)/2$, hence $|D_{\lceil H \rceil}| = n(n+1)/2$ and is not $O(|D_{\lceil G \rceil}|)$.

Hence the line graph transduction is neither in $\mathrm{MS}_{1,1}$ nor in $\mathrm{MS}_{2,2}$.

Example 1.46 (Transitive closure)

We have seen in Example 1.32 that the transitive closure : $G \mapsto G^+$ on directed graphs is an $\mathrm{MS}_{1,1}$-transduction. It is not an $\mathrm{MS}_{2,2}$-transduction: consider Q_n, the directed path with n vertices. We have $|D_{\lceil Q_n \rceil}| = 2n - 1$ and $|D_{\lceil Q_n^+ \rceil}| = n(n+1)/2$, hence we do not have $|D_{\lceil Q_n^+ \rceil}| = O(|D_{\lceil Q_n \rceil}|)$. Consequently the transitive closure of directed graphs is not an $\mathrm{MS}_{2,2}$-transduction.

Example 1.47 (Edge subdivision)

For G simple, directed and loop-free, we let $Sub(G)$ be the graph of the same type such that:

$$\begin{array}{lll} V_{Sub(G)} & := & V_G \cup E_G, \\ E_{Sub(G)} & := & \{(u,e),(e,v) \mid (e,u) \in in_{1G},\ (e,v) \in in_{2G}\}. \end{array}$$

This transformation, called *edge subdivision* consists in replacing directed edges by directed paths of length 2. We have $|V_{Sub(G)}| = |V_G| + |E_G|$ and $|E_{Sub(G)}| = 2|E_G|$. Edge subdivision is an $\mathrm{MS}_{2,2}$-transduction because the transformation of $\lceil G \rceil$ into $\lceil Sub(G) \rceil$ is a 3-copying monadic second-order transduction: a vertex

v of G is made into a vertex $(v,1)$ of $Sub(G)$, and an edge e linking u to v is made into a vertex $(e,1)$ and into edges $(e,2)$ and $(e,3)$ of $Sub(G)$ that link respectively $(u,1)$ to $(e,1)$ and $(e,1)$ to $(v,1)$. Its definition scheme will be given in Chapter 7 (Example 7.44). Since $|V_{Sub(G)}|$ is not $O(|V_G|)$, edge subdivision is not an $\mathrm{MS}_{1,1}$-transduction.

Example 1.48 (Identity)
The identity mapping is trivially an $\mathrm{MS}_{1,1}$- and an $\mathrm{MS}_{2,2}$-transduction, and hence also an $\mathrm{MS}_{2,1}$-transduction. It is not an $\mathrm{MS}_{1,2}$-transduction, as a clear consequence of Fact 1.37. However we have the following theorem which entails Theorem 1.44 with the help of Theorem 1.40.

Theorem 1.49 On each set of simple graphs that are planar, or of degree at most k, or of tree-width at most k for some fixed k, the identity is an $\mathrm{MS}_{1,2}$-transduction. □

We conclude this discussion with a diagram (Table 1.1) relating the different types of MS-transductions, where lines indicate strict inclusions from bottom-up. All inclusions are clear from the definitions and the above observations. That they are strict and that $\mathrm{MS}_{1,1}$ and $\mathrm{MS}_{2,2}$ are incomparable is proved by Examples 1.45, 1.46, 1.47 and 1.48.

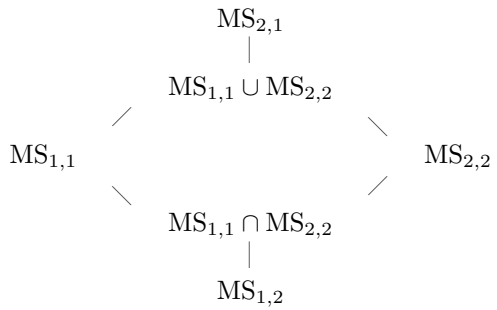

Table 1.1: The different classes of monadic second-order transductions.

The results in Section 1.7.3 are stated for $\mathrm{MS}_{1,1}$-transductions and the VR algebra. There are analogous statements for $\mathrm{MS}_{2,2}$-transductions and the HR algebra. We first observe a technical point: since the identity on trees or terms is an $\mathrm{MS}_{1,2}$-transduction and since the composition of two MS-transductions is an MS-transduction, a transduction from trees or terms to graphs is an $\mathrm{MS}_{1,j}$-transduction if and only if it is an $\mathrm{MS}_{2,j}$-transduction. There are thus only two types of transductions taking trees or terms as input to consider and we get the following fully similar results:

Theorem 1.50 For every k, the mapping that associates with a term t in $T(F^{\mathrm{HR}}_{[k]})$ the s-graph $t_{\mathbb{JS}}$ is an $\mathrm{MS}_{1,2}$-transduction. □

Theorem 1.51 (Equationality Theorem for the HR algebra)
A set of graphs is HR-equational if and only if it is the image of the set of binary rooted trees under an $MS_{1,2}$-transduction. □

As in Section 1.7.3, this implies, using the closure of monadic second-order transductions under composition (Theorem 1.39), that the image of an HR-equational set under an $MS_{2,2}$-transduction is again HR-equational (generalizing the logical version of the Filtering Theorem for HR). Moreover, using the Equationality Theorems for both the VR and the HR algebra, we obtain that the image of an HR-equational set under an $MS_{2,1}$-transduction is VR-equational, and the image of a VR-equational set under an $MS_{1,2}$-transduction is HR-equational. Note that if f is an $MS_{i,j}$-transduction and g is an $MS_{j,k}$-transduction $(i, j, k = 1, 2)$, then their composition $f \cdot g$ is an $MS_{i,k}$-transduction.

Using Proposition 1.20, Theorem 1.51 implies the following corollary.

Corollary 1.52 A set of graphs has bounded tree-width if and only if it is included in the image of the set of binary rooted trees under an $MS_{1,2}$-transduction. □

As for Theorem 1.42 and its corollary, both Theorem 1.51 and Corollary 1.52 hold with "the set of binary rooted trees" replaced by "the set of trees" or by "$T(F)$ where F is any finite signature with at least one constant symbol and at least one symbol of arity at least 2". In these results, graphs are without ports or sources in order to have simpler statements. Slightly more general results will be stated in Chapter 7.

From Corollaries 1.52 and 1.43 we get the following corollary from which quick proofs that certain sets of graphs have bounded or unbounded tree-width or clique-width can be obtained.

Corollary 1.53

(1) The image of a set of graphs of bounded tree-width under an $MS_{2,2}$-transduction (resp. under an $MS_{2,1}$-transduction) has bounded tree-width (resp. bounded clique-width).

(2) The image of a set of graphs of bounded clique-width under an $MS_{1,1}$-transduction (resp. under an $MS_{1,2}$-transduction) has bounded clique-width (resp. bounded tree-width).

(3) A set of simple planar graphs or of simple graphs of bounded degree has bounded tree-width if and only if it has bounded clique-width. □

Statements (1) and (2) follow from Corollaries 1.52 and 1.43, by Theorem 1.39. Since all proofs are effective, the new bound can be computed from the given bound and the definition scheme of the MS-transduction. Using again Theorem 1.39, statement (3) follows from Theorem 1.49 and the fact that the identity is an $MS_{2,1}$-transduction (Example 1.48). We will give in Section 2.5.5 a proof

of statement (3) that does not use transductions and gives a good estimate of the bound on tree-width.

To conclude this section, let us stress the nice parallelism between two groups of definitions:

(1) the VR algebra, clique-width and MS formulas, and

(2) the HR algebra, tree-width and MS_2 formulas.

The Recognizability Theorem and the Equationality Theorem have fully analogous statements for both groups. Furthermore, the same graph theoretic conditions, those of Theorem 1.44 (and more general ones), ensure the equivalence of clique-width and tree-width, and simultaneously of MS and MS_2 formulas. The $MS_{1,2}$- and $MS_{2,1}$-transductions define "bridges" between the "world of bounded clique-width" and that of "bounded tree-width". The facts show how intimate are the relationships between logical and combinatorial notions.

1.9 Relational structures

Terms, graphs, labelled graphs, hypergraphs of different types are, or rather can be conveniently represented by *relational structures*. We have only seen up to now relational structures with unary and binary relations, that correspond to vertex- and edge-labelled graphs. However, many of our results can be proved without any difficulty for general relational structures.

In order to illustrate the usefulness of relational structures in Discrete Mathematics, we will present the examples of *betweenness relations* and *cyclic orderings*, two combinatorial notions defined in a natural way as ternary relations. Furthermore, in a different domain, the theory of *relational databases* is based on the concept of relational structure (see the book by Abiteboul, Hull and Vianu [*AbiHV]). However, our theory will not bring much to this field for reasons that we will discuss briefly.

1.9.1 Relational signatures and structures

A *relational signature* (to be contrasted with the notion of a functional signature defined in Section 1.1.4) is a finite set $\mathcal{R}$ of *relation symbols* where each symbol R of $\mathcal{R}$ has an associated arity $\rho(R)$ in $\mathcal{N}_+ := \mathcal{N} - \{0\}$. A *relational structure* of type $\mathcal{R}$, called simply an $\mathcal{R}$-*structure*, is a tuple $S = \langle D_S, (R_S)_{R\in\mathcal{R}}\rangle$ consisting of a finite (possibly empty) *domain* D_S and of a $\rho(R)$-ary relation[43] R_S for each $R \in \mathcal{R}$. The set of $\mathcal{R}$-structures is denoted by $STR(\mathcal{R})$. We let $\rho(\mathcal{R}) := \max\{\rho(R) \mid R \in \mathcal{R}\}$. A signature $\mathcal{R}$ (resp. an $\mathcal{R}$-structure) is *binary* if $\rho(\mathcal{R}) \leq 2$.

Since every k-ary function can be considered as a $(k+1)$-ary relation, there is no loss of generality in considering relational structures as opposed to more

[43] A k-ary relation can be defined as a subset of D_S^k or, equivalently, as a total function: $D_S^k \to \{True, False\}$.

general logical structures containing also functions. Although a *constant*, i.e., a 0-ary function can be replaced by a (singleton) unary relation, it will be convenient (for instance for representing the sources of graphs) to allow constants. However, in this introductory section, we will only consider relational structures without constants.

Formulas are written with atomic formulas of the two forms $x_1 = x_2$ and $R(x_1, \ldots, x_{\rho(R)})$ where $x_1, \ldots, x_{\rho(R)}$ are individual variables. The notion of an *MS-expressible* property of $\mathcal{R}$-structures follows immediately. A subset L of $STR(\mathcal{R})$, the set of all $\mathcal{R}$-structures, is *MS-definable* if it is the set of finite models of a monadic second-order sentence φ, formally, if $L = \{S \in STR(\mathcal{R}) \mid S \models \varphi\}$.

For expressing graph properties by monadic second-order formulas, we have defined two relational structures associated with a graph G, denoted by $\lfloor G \rfloor$ and $\lceil G \rceil$. We have observed that certain graph properties are monadic second-order expressible *via* the "rich" representation $\lceil G \rceil$, but not *via* the "natural" one $\lfloor G \rfloor$. The former properties are called MS_2-expressible. A similar extension of monadic second-order logic can be defined for relational structures. We let $\mathcal{R}^{Inc} := \mathcal{R} \cup \{in_i \mid 1 \leq i \leq \rho(\mathcal{R})\}$ with $\rho(R) = 1$ for $R \in \mathcal{R}$ and $\rho(in_i) = 2$ for $i = 1, \ldots, \rho(\mathcal{R})$. The *incidence structure* of $S = \langle D_S, (R_S)_{R \in \mathcal{R}} \rangle$ is the $\mathcal{R}^{Inc}$-structure $Inc(S)$ defined as

$$\langle D_S \cup T_S, (R_{Inc(S)})_{R \in \mathcal{R}}, in_{1\,Inc(S)}, \ldots, in_{k\,Inc(S)} \rangle$$

where $k := \rho(\mathcal{R})$ and:

$$T_S := \{(R, d_1, \ldots, d_{\rho(R)}) \mid R \in \mathcal{R},\ (d_1, \ldots, d_{\rho(R)}) \in R_S\},$$

$$R_{Inc(S)}(d) :\Longleftrightarrow d = (R, d_1, \ldots, d_{\rho(R)}) \in T_S \text{ for some } d_1, \ldots, d_{\rho(R)} \in D_S,$$

$$in_{i\,Inc(S)}(d, d') :\Longleftrightarrow d \in T_S,\ d' \in D_S \text{ and } d = (R, d_1, \ldots, d_{\rho(R)}) \text{ for some } R \in \mathcal{R} \text{ and } d_1, \ldots, d_{\rho(R)} \text{ such that } d' = d_i.$$

It is clear that two $\mathcal{R}$-structures S and S' are isomorphic if and only if $Inc(S)$ and $Inc(S')$ are isomorphic. The incidence structure $Inc(S)$ of S is actually a vertex- and edge-labelled bipartite directed graph. Furthermore, each relation $in_{i\,Inc(S)}$ is functional. A set of $\mathcal{R}$-structures is *MS_2-definable* if it is $\{S \in STR(\mathcal{R}) \mid Inc(S) \models \varphi\}$ for a monadic second-order sentence φ over the signature $\mathcal{R}^{Inc}$. As for graphs, we obtain the notion of an *MS_2-expressible* property of $\mathcal{R}$-structures by replacing S by $Inc(S)$.

Since the incidence structure of a relational structure is a labelled graph, the results concerning MS_2 formulas and labelled graphs of bounded tree-width transfer easily to relational structures. It is not difficult to see that the identity on incidence structures (of $\mathcal{R}$-structures) is an $\mathrm{MS}_{1,2}$-transduction, which implies that $Inc(S)$ has the same MS_2-expressible and MS_1-expressible properties (cf. Theorems 1.49 and 1.44).

We define $twd^{Inc}(S) := twd(Inc(S))$ to be used as parameter. For each k, we define $STR_k(\mathcal{R})$ as the class $\{S \in STR(\mathcal{R}) \mid twd^{Inc}(S) \leq k\}$ and we get the following result.

Theorem 1.54 Let $\mathcal{R}$ be a relational signature.

(1) The model-checking problem for CMS_2 sentences and the class of $\mathcal{R}$-structures is fixed-parameter linear with respect to $twd^{Inc}(S) + |\varphi|$ where S is the input structure and φ is the input sentence.

(2) For each $k \in \mathcal{N}$, the CMS_2-satisfiability problem for the class $STR_k(\mathcal{R})$ is decidable.

(3) If a subset of $STR(\mathcal{R})$ has a decidable MS_2-satisfiability problem, then it is contained in $STR_m(\mathcal{R})$ for some m. □

This theorem generalizes to $\mathcal{R}$-structures the parts of Theorems 1.25, 1.26 and 1.29 that concern tree-width and CMS_2-expressible graph properties. Establishing analogous results for CMS properties (as opposed to CMS_2 properties) raises difficult open problems.

1.9.2 Betweenness and cyclic ordering

We now present the two combinatorial notions of betweenness and cyclic ordering that are naturally defined as ternary relations. They raise open questions relative to monadic second-order expressibility. All results stated below will be proved in Section 9.1.

With a finite linear order $\langle D, \leq \rangle$ such that $|D| \geq 3$ we associate the following ternary relation, called its *betweenness relation*:

$$B(x, y, z) \ :\Longleftrightarrow \ (x < y < z) \vee (z < y < x)$$

(where $x < y$ means "$x \leq y$ and $x \neq y$"). We denote it by $B(\leq)$. This relation satisfies the following properties, for all $x, y, z, t \in D$:

(B1) $B(x, y, z) \Rightarrow x \neq y \wedge x \neq z \wedge y \neq z$,

(B2) $B(x, y, z) \Rightarrow B(z, y, x)$,

(B3) $B(x, y, z) \Rightarrow \neg B(y, z, x)$,

(B4) $B(x, y, z) \wedge B(y, z, t) \Rightarrow B(x, y, t) \wedge B(x, z, t)$,

(B5) $B(x, y, z) \wedge B(y, t, z) \Rightarrow B(x, y, t) \wedge B(x, t, z)$,

(B6) $x \neq y \wedge x \neq z \wedge y \neq z \Rightarrow B(x, y, z) \vee B(y, z, x) \vee B(z, x, y)$.

Conversely, if B is a ternary relation satisfying these properties, it is $B(\leq)$ for some linear order on D, hence is a betweenness relation. A set $X \subseteq D^3$ is *consistent for betweenness* if $X \subseteq B$ for some betweenness relation B on D. The problem Betweenness consisting in deciding whether a given set $X \subseteq D^3$ is consistent for betweenness is NP-complete ([*GarJoh]).

If X is consistent for betweenness, we define

$$\widehat{X} := \bigcap \{B \mid X \subseteq B, \ B \text{ is a betweenness relation}\}.$$

The set $\widehat{X}$ satisfies properties (B1)-(B5). We say that X is a *partial betweenness relation on* D if $\widehat{X} = X$. These definitions raise the following open questions, where a ternary relation $X \subseteq D^3$ is identified with the $\mathcal{R}$-structure $\langle D, X\rangle$ (for some fixed singleton $\mathcal{R}$):

Questions 1.55
(1) Is the set of relational structures $\{\langle D, X\rangle \mid X \text{ is consistent for betweenness}\}$ MS-definable?
(2) Is the set of partial betweenness relations MS-definable?

For MS_2 the answers to these questions are positive.

Proposition 1.56 The set of partial betweenness relations and the set of ternary relations that are consistent for betweenness are MS_2-definable. □

For $X \subseteq D^3$, we define the *size* of $\langle D, X\rangle$ as $|D| + |X|$ and $twd^{Inc}(X)$ as the tree-width of the labelled graph $Inc(\langle D, X\rangle)$. From Proposition 1.56 and Theorem 1.54(1) we immediately obtain the next result.

Corollary 1.57 The problem BETWEENNESS is fixed-parameter linear with respect to twd^{Inc}. □

We now consider the similar notion of cyclic ordering. With a finite linear order $\langle D, \leq\rangle$ such that $|D| \geq 3$, we associate the ternary relation:

$$C(x, y, z) \ :\Longleftrightarrow \ (x < y < z) \vee (y < z < x) \vee (z < x < y).$$

Let $D := \{d_1, \ldots, d_n\}$ with $d_1 < d_2 < \cdots < d_n$ where $d_1, \ldots, d_n$ are points on a circle such that, according to some orientation of the plane, d_{i+1} follows d_i and d_1 follows d_n. Then $C(x, y, z)$ expresses that, if one traverses the circle according to this orientation by starting at x, one meets y before z. We denote by $C(\leq)$ the ternary relation associated with $\leq$ in this way. A relation of this form is a *cyclic ordering*. A cyclic ordering C satisfies the following properties, for every x, y, z, t of its domain D:

(C1) $C(x, y, z) \Rightarrow x \neq y \wedge x \neq z \wedge y \neq z$,

(C2) $C(x, y, z) \Rightarrow C(y, z, x)$,

(C3) $C(x, y, z) \Rightarrow \neg C(x, z, y)$,

(C4) $C(x, y, z) \wedge C(y, t, z) \Rightarrow C(x, y, t) \wedge C(x, t, z)$,

(C5) $x \neq y \wedge y \neq z \wedge x \neq z \Rightarrow C(x, y, z) \vee C(x, z, y)$.

Every ternary relation satisfying (C1)-(C5) is a cyclic ordering. A subset X of D^3 is *consistent for cyclic ordering* if $X \subseteq C$ for some cyclic ordering on D. The problem CYCLIC ORDERING that consists in deciding if a set $X \subseteq D^3$ is consistent for cyclic ordering is NP-complete ([GalMeg]).

As for betweenness, for $X \subseteq D^3$, we let $\widehat{X}$ be the intersection of all cyclic orderings C on D such that $X \subseteq C$ (and $\widehat{X}$ is undefined if $X = \emptyset$ or if there is no cyclic ordering containing X). A *partial cyclic ordering* on a set D is defined as a subset of D^3 such that $\widehat{X} = X$. For cyclic ordering, we have the same results and open questions as for betweenness.

1.9.3 Relational databases

The theory of relational databases (exposed in the book [*AbiHV]) is based on relational structures. In this theory, a relational signature $\mathcal{R}$ is called a *database schema*, its elements are called *relation schemas*, and an $\mathcal{R}$-structure is called a *database instance*. A *query* is a syntactic or algorithmic description of a relation with specified arity written in some *query language* and defined in terms of the relations stored in the considered database instance. One concern is to compare the expressive powers of several such languages. Another one is to construct efficient algorithms for evaluating these relations, that is, to list their tuples or sometimes, only to count them.

Theorem 1.54 yields linear-time algorithms for such computations (and for fixed queries) in cases where the input structures are constrained to belong to $STR_k(\mathcal{R})$ for some fixed k, or to satisfy some similar condition (e.g., for binary structures, to have bounded clique-width), and formulas are required to be monadic second-order. In the case of databases, there is usually no reason to assume that the relational structure modelling the database instance satisfies such constraints. Constraints are rather put on the formulas expressing queries in order to ensure the existence of efficient algorithms. These constraints are formulated in terms of tree-width and *hypertree-width* of certain graphs associated with formulas: we refer the reader to the comprehensive article by Gottlob *et al.* [GotLS]. Hence, the basic concepts of relational structures and logical formulas are the same as in the algorithms of Section 1.5, but the methods for constructing fixed-parameter tractable algorithms are not.

1.10 References

The collective book [*Com+] by Comon *et al.*, readable online, is a thorough study of finite automata on terms. Another reference is the book chapter [*GecSte].

Graph grammars defined in terms of graph rewritings are surveyed in two chapters ([*EngRoz, *DreKH]) of the first volume of the handbook of graph grammars and graph transformations [*Roz] edited by Rozenberg. Another similar survey is the book chapter [*Eng97]. Most of the material referred to in Sections 1.1-1.8 is surveyed in [*Cou97], another chapter of [*Roz].

The books by Diestel [*Die] and by Mohar and Thomassen [*MohaTho] are our main references for general graph theory and for graphs embedded on surfaces respectively.

The books by Downey and Fellows [*DowFel] and by Flum and Grohe [*FluGro] present in detail the theory of fixed-parameter tractability and contain important sections on tree-decompositions and their algorithmic applications. The surveys by Grohe [*Gro] and by Kreutzer [*Kre] focus on algorithms for problems expressed by first-order and monadic second-order sentences relative to graphs that are structured in various ways.

One of our objectives is to extend to finite graphs the algebraic view of Formal Language Theory initiated by Mezei and Wright [MezWri]. The least fixed-point characterization of context-free languages due to Ginsburg and Rice [GinRic] and to Chomsky and Schützenberger [ChoSch] has inspired the notion of equational sets, defined in [MezWri]. This article extends to general algebras the notion of recognizability studied for monoids by Eilenberg, Schützenberger and many others: see the books by Eilenberg [*Eil] and Sakarovitch [*Sak].

Monadic second-order logic on words, terms and trees, either finite or infinite, and its relationships with automata is a vast domain presented in the two book chapters by Thomas [*Tho90] and [*Tho97a]. From this theory, we will only use Theorem 1.16 by Doner [Don] and Thatcher and Wright [ThaWri] that generalizes to terms the corresponding basic result established for words by Büchi [Büc], Elgot [Elg] and Trakhtenbrot [Tra].

We will not study countable graphs and structures. For this rich topic we refer the reader to the book chapters by Thomas [*Tho90] and [*Tho97a], and to the books [*GräTW] and [*FluGräW].