Theoretical Computer Science 109 (1993) 49–82
Elsevier

Monadic second-order evaluations on tree-decomposable graphs*

B. Courcelle and M. Mosbah

*Université Bordeaux-I, Laboratoire d'Informatique**, 351, cours de la Libération, 33405 Talence, France*

Abstract

Courcelle, B. and M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theoretical Computer Science 109 (1993) 49–82.

Every graph generated by a hyperedge replacement graph-grammar can be represented by a tree, namely the derivation tree of the derivation sequence that produced it. Certain functions on graphs can be computed recursively on the derivation trees of these graphs. By using monadic second-order logic and semiring homomorphisms, we describe in a single formalism a large class of such functions. Polynomial and even linear algorithms can be constructed for some of these functions. We unify similar results obtained by Takamizawa et al. (1982), Bern et al. (1987), Arnborg et al. (1991) and Habel et al. (1989).

0. Introduction

Many $\mathcal{NP}$-complete problems become polynomial when restricted to particular sets of graphs. A number of such cases are discussed in [22]. More informative than isolated results are *metaresults*, exhibiting classes of sets of graphs, classes of problems having polynomial algorithms on the sets of graphs of the corresponding classes, and uniform descriptions of these algorithms. Such an approach is that of [25, 6, 3, 7, 8, 12].

Correspondence to: B. Courcelle, Université Bordeaux-I, Laboratoire d'Informatique, 351, cours de la Libération, 33405 Talence, France. Email addresses of the authors: courcell@geocub.greco-prog.fr and mosbah@geocub.greco-prog.fr.

*This work has been supported by the "Programme de Recherches Coordonnées: Mathématiques et Informatique" and the ESPRIT Basic Research Action 3299 "Computing by Graph Transformations". A preliminary version of this paper has been presented at the conference *WG '91*, Fischbachau, Germany, in June 1991, and has appeared in the proceedings (Lecture Notes in Computer Science, Vol. 570).

** Laboratoire associé au CNRS.

The present paper follows this line of research, where the notion of a problem is extended into that of an *evaluation*. An evaluation is a function that associates with every graph a value in some set S, say $\mathbb{N}, \mathbb{R}, \mathbb{N} \times \mathbb{N}$. Hence, a decision problem is an evaluation where $S = \{\textbf{true}, \textbf{false}\}$. However, the value of an evaluation can also be a set of vertices, or a set of edges of the given graph. It follows that we can consider the problem of evaluating *optimal subgraphs* of the given graphs, and we can consider the problems raised in [6], in particular, that of giving a syntactic characterization of the so-called regular properties. See the conclusion of [6].

The sets of graphs we deal with are those definable by *hyperedge replacement* grammars. The sets of series–parallel graphs, Halin graphs and outerplanar graphs are examples of such sets. So are, for each k, the set of graphs of tree-width at most k and the set of graphs of bandwidth at most k. Since every set of graphs generated by a hyperedge replacement grammar has bounded tree-width, the tree-width boundedness is common to all these cases. These grammars can generate sets of directed as well as undirected graphs, with possible labels attached to vertices and/or to edges. They can also generate sets of hypergraphs.

Grammars are essential in that every generated graph can be described by a tree, namely the derivation tree of the derivation producing the graph. Graph evaluations of the appropriate type (we shall say, following Habel [19], that they are *compatible* with the grammar) can be computed by means of one bottom-up traversal of the derivation tree (like in attribute grammars when there are only synthesized attributes.)

The purpose of this paper is to describe, in a uniform way, a class of compatible evaluations that is as large as possible. This will be done in a formalism that associates logic and algebra, or, more precisely, monadic second-order logic and semiring homomorphisms.

Let us present briefly and informally the basic ideas. Let φ be a monadic second-order formula with set W of free variables. For every graph G (we consider it as a logical structure), we denote by $\boldsymbol{sat}(\varphi)(G)$ the set of W-assignments in G that satisfy the formula φ. The fundamental result of Courcelle [14] says that $\boldsymbol{sat}(\varphi)(G)$ can be evaluated bottom-up on any derivation tree of G.

Some evaluations v can be expressed by $v(G) = h(\boldsymbol{sat}(\varphi)(G))$. This expression does not give immediately an efficient way of computing $v(G)$, because $\boldsymbol{sat}(\varphi)(G)$ is frequently a very large, although finite, set. If h is a homomorphism, in an appropriate sense, the mapping $v(G)$ can be evaluated *directly* bottom-up on any derivation tree of G, without needing the costly computation of $\boldsymbol{sat}(\varphi)(G)$. Linear algorithms can thus be constructed, provided the computations to be done at every node of the tree take constant time.

We obtain in this way a new proof of some results established by Arnborg et al. [3], and linear algorithms[1] in some cases not covered by the *extended monadic second-order logic* introduced in their paper (for instance, when one wishes to compute the sum of cardinalities of all sets satisfying an MS-formula with one free set variable). We

[1] For uniform cost measure, as everywhere else in this paper.

obtain a syntactic expression for a large class of *compatible* evaluations, including the special cases considered in [19, 20]. We also obtain linear or polynomial algorithms for the sample optimization problems considered in [6]. We actually apply the method introduced in this paper to a large class of problems, described syntactically in a uniform way, and we answer the questions raised in its conclusion.

However, we do not claim to cover in our formalism *all* graph problems that are polynomial, say, on partial k-trees (for fixed k), i.e. on simple loop-free undirected graphs of tree-width at most k. Bodlaender [9] has considered games on graphs. For one of them, called VERTEX GENERALIZED GEOGRAPHY, the existence of a winning strategy for one of the two players is decidable in linear time on partial k-trees, but does not seem to be expressible in our formalism. Similarly, the diameter of a partial k-tree or its chromatic index [10] can be computed in polynomial time, but we cannot express them in our syntactic framework.

Two closely related papers are [12] and [21]. The former uses monadic second-order logic to specify decision problems and evaluations (like the one counting the number of tuples satisfying a given monadic second-order formula), and to obtain linear algorithms. The latter uses semiring homomorphisms for handling evaluations and constructing linear or polynomial algorithms; however, decision problems and evaluations are specified informally, i.e. outside any syntactic framework like the ones used in [3, 12] and in the present paper.

We also establish that if a graph transformation is specified by monadic second-order formulas, i.e. if it is a *definable transduction* as introduced in Courcelle [15], then it is computable in polynomial time for input graphs or hypergraphs of bounded tree-width.

The paper is organized as follows. Section 1 gives a few definitions concerning many-sorted algebras, graphs and graph operations. Section 2 introduces monadic second-order logic, monadic second-order evaluations on graphs, and establishes the central results of this paper (Theorems 2.3 and 2.10). Section 3 discusses the construction of efficient algorithms. Section 4 reviews the main evaluation structures and contains the applications. Section 5 compares our approach with that of [20] and raises a few open questions.

1. Notations and definitions

In the following definitions, we let $\mathcal{S}$ be a possibly infinite set of sorts, F an $\mathcal{S}$-signature, and $M = \langle (M_s)_{s\in\mathcal{S}}, (f_M)_{f\in F}\rangle$ an F-algebra. In particular, if $f\in F$ has profile $s_1 \times s_2 \times \cdots \times s_n \to s$, then f_M is a total mapping $M_{s_1} \times M_{s_2} \times \cdots \times M_{s_n} \to M_s$. The functions f_M are called the *operations* of M. A *derived operation* of M is a function $M_{s_1} \times M_{s_2} \times \cdots \times M_{s_n} \to M_s$, defined by a term t of sort s built with the operation symbols from F, variables $x_1, \ldots, x_n$ of respective sorts $s_1, \ldots, s_n$ and additional constants denoting fixed elements of M. We assume that each variable has at most one occurrence in t, and that t is not reduced to a single variable (we would obtain in this

case the identity). For more details, the reader is referred to [14, 13, 5]. If X is a set of variables with sorts, we denote by $M(F, X)$ the set of finite well-formed terms written with F and X.

The set of subsets of a set D is denoted by $\mathscr{P}(D)$. Its set of finite subsets is denoted by $\mathscr{P}_f(D)$.

1.1. Computable evaluations

Definition 1.1 (*Inductive families of evaluations*). Let D be any set. A *family of evaluations* on M is a set $\mathscr{E}$ of unary mappings such that each mapping v in $\mathscr{E}$ maps $M_{\alpha(v)}$ into D. The object $\alpha(v)$ belongs to $\mathscr{S}$ and is called the *type* of v.

A family of evaluations $\mathscr{E}$ is *F-inductive* if for every f in F of profile $s_1 \times s_2 \times \cdots \times s_n \to s$, for every v in $\mathscr{E}$ of type s, there exists an operator $\theta_{v,f}$ on D and a sequence $(v_{1,1}, \ldots, v_{1,m_1}, \ldots, v_{2,1}, \ldots, v_{2,m_2}, \ldots, v_{n,m_n})$ of length $(m_1 + m_2 + \cdots + m_n)$ of elements of $\mathscr{E}$ such that

(1) $\alpha(v_{i,j}) = s_i$ for all $j = 1, \ldots, m_i$,

(2) for all $d_1 \in M_{s_1}, \ldots, d_n \in M_{s_n}$,

$$v(f_M(d_1, \ldots, d_n)) = \theta_{v,f}(v_{1,1}(d_1), \ldots, v_{1,m_1}(d_1), v_{2,1}(d_2), \ldots, v_{2,m_2}(d_2), \ldots, v_{n,m_n}(d_n)).$$

The sequence $(\theta_{v,f}, v_{1,1}, \ldots, v_{n,m_n})$ is called a *decomposition* of v with respect to f, and $\theta_{v,f}$ its *decomposition operator*.

The existence of such a decomposition means that the value of v for any object of the form $f_M(d_1, \ldots, d_n)$ can be determined and computed from the values of finitely many mappings of $\mathscr{E}$ at $d_1, \ldots, d_n$.

If $D = \{\textbf{true}, \textbf{false}\}$, then a family of evaluations is a set of predicates and an operator $\theta_{v,f}$ is a Boolean expression. This special case has been considered in [14].

Lemma 1.2. *If $\mathscr{E}$ is F-inductive, then it is G-inductive, where G is any set of derived operations constructed over F.*

Proof. Let g be a derived operation, defined by a term t written with some symbols of F, some variables and some constants. For every v of type $\sigma(t)$, one can construct a decomposition of v with respect to g from those of finitely many mappings from $\mathscr{E}$, with respect to the functions of F occurring in t. This construction uses also some values of some functions of $\mathscr{E}$ for the constants occurring in t. The usefulness of derived operations is discussed in Section 3.4. □

Definition 1.3 (*Inductively computable evaluations*). Let $s_0 \in \mathscr{S}$ be a sort of interest. A mapping $v: M_{s_0} \to D$ is *F-inductively computable* if there exists a family of evaluations $\mathscr{E}$ such that

(1) $v \in \mathscr{E}$,
(2) $\mathscr{E}$ has finitely many functions of each type and
(3) $\mathscr{E}$ is F-inductive, with known decompositions.

Hence, by Lemma 1.2, if v is F-inductively computable, then it is G-inductively computable for every set G of derived operations over F.

We shall use these definitions in the case where $M_{\cdot}$ is the H_A-algebra of finite hypergraphs defined in [5] and the evaluations are functions on graphs defined in *monadic second-order logic*, as explained below.

1.2. Graphs and hypergraphs

Hypergraphs are finite, labeled, directed and are equipped with a sequence of distinguished vertices called the *sources*, as defined in [14, 13, 5]. The labels are chosen in a ranked alphabet A, i.e. an alphabet given with a mapping $\tau: A \to \mathbb{N}$. We shall call $\tau(a)$ the *type* of a. The type of the label of a hyperedge must be equal to the length of its sequence of vertices.

Formally, a *k-hypergraph*, also called a *hypergraph of type k* over A, is a quintuple $G = \langle V_G, E_G, lab_G, vert_G, src_G \rangle$, where

- V_G is the finite set of vertices,
- E_G is the finite set of hyperedges (with $V_G \cap E_G = \emptyset$),
- $lab_G: E_G \to A$ is the hyperedge labeling function,
- $vert_G: E_G \to V_G^*$ associates with every hyperedge the sequence of its vertices; the ith element of $vert_G(e)$ will be denoted by $vert_G(e, i)$,
- src_G is a sequence of k vertices called the *sources*; $src_G(i)$ denotes the ith element of this sequence; we shall consider src_G as a mapping $[k] \to V_G$, where $[k]$ denotes $\{1, 2, \ldots, k\}$ (and $[0] = \emptyset$).

We denote by $\mathbf{G}_k(A)$ the set of all finite hypergraphs of type k over A. If $\tau(a) = 2$ for each $a \in A$, then a hypergraph over A is just a *graph*. In order to simplify the terminology, we shall formulate most of our results and definitions for graphs. Nevertheless, the extension to hypergraphs is straightforward. (The reader may also consider that we use the term "*graph*" as an abbreviation for "*hypergraph*", and the term "*edge*" as an abbreviation for "*hyperedge*".)

In contrast to what was done in [5, 13 15], we do not consider two isomorphic graphs as equal. The reason is that we aim at results concerning graph algorithms, and it is convenient to view vertices and edges as concrete objects. We shall consider that the sets V_G and E_G are subsets of a fixed countable linearly ordered set D.

1.3. Graph operations

A graph operation is a mapping that associates a graph with one or several graphs. Graph operations are also essential in [6, 12, 25], and in other works dealing with graph algorithms (see [2]). The following three basic graph operations have been defined in [5, 14, 13].

First, if G' is a k'-graph and G'' is a k''-graph *disjoint with* G', i.e. such that $(V_{G'} \cup E_{G'}) \cap (V_{G''} \cup E_{G''}) = \emptyset$, then $G' \oplus G''$ denotes their *union*, equipped with the concatenation of $src_{G'}$ and $src_{G''}$ as a sequence of sources. Hence, $\oplus$ is a partial operation.

Second, if $1 \leqslant i < j \leqslant n$, we let $\theta_{i,j}$ be the mapping such that if G is an n-graph, then $G' = \theta_{i,j}(G)$ is obtained by "*fusing*" the ith and the jth sources of G. If $src_G(i) = src_G(j)$, then $G' = G$. Otherwise, $V_{G'} = V_G - \{src_G(j)\}$, $E_{G'} = E_G$ and the vertex $src_G(j)$ is replaced by $src_G(i)$ everywhere in the mapping $vert_G$ and in the sequence src_G.

Finally, if $\alpha : [p] \to [n]$ is a total mapping, if G is an n-graph, then $\sigma_\alpha(G)$ is the p-graph consisting of G equipped with $src_G(\alpha(1)), src_G(\alpha(2)), \ldots, src_G(\alpha(p))$ as a sequence of sources, instead of src_G. Note that $\theta_{i,j}$ and σ_α are total.

The set of nonnegative integers $\mathbb{N}$ is the set of sorts of the signature H consisting of

$\oplus_{n,m}$ of profile $n \times m \to n+m$,

$\theta_{i,j,n}$ of profile $n \to n$ and

$\sigma_{\alpha,p,n}$ of profile $n \to p$.

We shall also use a constant a of sort $\tau(a)$ for each $a \in A$, the constants **1** and **0** of respective sorts 1 and 0. We let

$$H_A = H \cup A \cup \{\mathbf{0}, \mathbf{1}\}.$$

We say that a term t in $M(H_A)$ denotes a graph G if

- either $t = a$ and G is a $\tau(a)$-graph consisting of one edge e with label a and such that $src_G = vert_G(e)$,
- or $t = \mathbf{0}$ and G if the empty 0-graph,
- or $t = \mathbf{1}$ and G has a unique vertex that is the unique source and no edge,
- or $t = t_1 \oplus_{n,m} t_2$ and $G = G_1 \oplus G_2$, where t_1 denotes G_1 and t_2 denotes G_2, of types n and m, respectively,
- or $t = \theta_{i,j,n}(t_1)$ and $G = \theta_{i,j}(G_1)$, where t_1 denotes G_1 of type n,
- or $t = \sigma_{\alpha,p,n}(t_1)$ and $G = \sigma_\alpha(G_1)$, where t_1 denotes G_1 of type n.

It follows from this definition that a term t in $M(H)$ denotes several graphs which are all isomorphic. We shall write $G = \mathbf{val}(t)$ as an abbreviation of "t denotes G".

The signature H_A is infinite. We shall get effective results and efficient algorithms by restricting ourselves to graphs that are denoted by terms over finite subsets of H_A with finite subsets of $\mathbb{N}$ as sets of sorts, and finitely many operations.

We quote from [5], [2], and [13] the following results.

Fact. (1) *A set* $L \subseteq \mathbf{G}_n(A)$ *is expressible by finitely many of the operations of* H_A *iff it has bounded tree-width.*

(2) *This is the case of partial k-trees (i.e. of simple loop-free undirected graphs of tree-width at most k), sets of graphs and hypergraphs generated by hyperedge replacement grammars, and k-terminal recursive families of graphs in the sense of Wimer* [28].

We shall, in general, omit the sort subscripts and denote, slightly ambiguously, the above operations by $\oplus$, $\theta_{i,j}$ or σ_α.

2. Monadic second-order evaluations on graphs

By considering a graph as a logical structure, we can express graph properties in logic. In the present paper, we consider graph properties (and more generally graph evaluations) expressible in *counting* monadic second-order logic.

Definition 2.1 (*Hypergraphs as logical structures*). In order to express properties of k-hypergraphs over A, we define the following symbols:

v, the *vertex* sort,

e, the *edge* sort,

$\mathbf{s}_i$, a constant of sort **v**, for each i, $1 \leqslant i \leqslant k$,

$\mathbf{edg}_a$, a predicate symbol of arity **evv**...**v** (with $\tau(a)$ occurrences of **v**), for each a, $a \in A$.

With a k-hypergraph G over A, we associate the logical structure $|G| = \langle V_G, E_G, (\mathbf{s}_{iG})_{i\in[k]}, (\mathbf{edg}_{aG})_{a\in A}\rangle$, where V_G is the domain of sort **v**, E_G is the domain of sort **e**, $\mathbf{s}_{iG}$ is the ith source of G, and $\mathbf{edg}_{aG}(e, v_1, \ldots, v_n) = \mathit{true}$ iff $lab_G(e) = a$ and $vert_G(e) = (v_1, \ldots, v_n)$.

Definition 2.2 (*Counting monadic second-order logic*). To build formulas, we use object variables $u, x, y, z, u', \ldots$ of sort **v** or **e**, denoting, respectively, vertices or edges, and set variables U, X, Y, Z, U' of sort **v** or **e**, denoting, respectively, sets of vertices or sets of edges.

Let $\mathcal{W}$ be a finite sorted set of variables $\{u, u', \ldots, U, U', \ldots\}$, each of them having a sort $\sigma(u), \sigma(u'), \ldots, \sigma(U), \sigma(U'), \ldots$ in $\{\mathbf{v}, \mathbf{e}\}$. We denote by $\mathcal{W}_s$ the set $\mathcal{W} \cup \{\mathbf{s}_1, \ldots, \mathbf{s}_k\}$. Uppercase letters denote set variables and lowercase letters denote object variables or constants.

The set of atomic formulas consists of

$u = u'$, with $u, u' \in \mathcal{W}_s$, $\sigma(u) = \sigma(u')$,

$u \in U$, with $u, U \in \mathcal{W}_s$, $\sigma(u) = \sigma(U)$,

$\mathbf{edg}_a(u, u'_1, \ldots, u'_n)$, with $u, u'_1, \ldots, u'_n \in \mathcal{W}_s$, $\sigma(u) = \mathbf{e}$, $\sigma(u'_1) = \cdots = \sigma(u'_n) = \mathbf{v}$,

$\mathbf{Card}_{m,p}(U)$, with $0 \leqslant m < p$ and $2 \leqslant p$.

The last formula has the following meaning:

$\mathbf{Card}_{m,p}(U) = \mathbf{true}$ iff $\mathbf{Card}(U) = m \bmod p$ (where $\mathbf{Card}(U)$ denotes the cardinality of U).

The language of *counting monadic second-order logic* (CMS) is the set of formulas formed with the above atomic formulas together with the Boolean connectives and quantifications over object and set variables. The language of monadic second-order logic (MS) is the set of such formulas not using the atomic formulas $\mathbf{Card}_{m,p}(U)$. It has been proved by Courcelle [14] that the former language is more powerful than the latter.

In what follows, we consider CMS-formulas with set variables only. This is not a loss of generality because each CMS-formula can be translated into an equivalent CMS-formula using only set variables (see [26] or [14] for details). We denote by $\Phi^{h,q}_{A,k}(\mathcal{W})$ the set of CMS-formulas of height at most h and variables in $\mathcal{W}$, where $p \leqslant q$ in all subformulas of the form $\mathbf{Card}_{m,p}(U)$. The integer k is a bound on the types of the graphs these formulas express properties of, and A is the finite set of edge labels. The height of a formula is the depth of nested quantifications.

To shorten our writing, we will fix $\mathcal{W}, h, q$ and A and refer to the previous sets by Φ_k. We shall work with several types of graphs at the same time; hence, k will have to vary. For every graph G, D_G will denote the set $E_G \cup V_G$. (We take always $E_G \cap V_G = \emptyset$.) Furthermore, we shall let D be a countably infinite set (say the set of integers) such that $D_G \subseteq D$ for all graphs G.

A formula, say φ, will usually be given as a member of $\Phi_k(\mathcal{W})$, where $\mathcal{W} = \{X_1, \ldots, X_n\}$. We shall denote it also by $\varphi(X_1, \ldots, X_n)$ in order to recall what is in $\mathcal{W}$. This does not mean that each variable $X_1, \ldots, X_n$ actually occurs free in φ, but only that all free variables are in $\{X_1, \ldots, X_n\}$. A $\mathcal{W}$-assignment in G is a mapping ν associating with every variable X in $\mathcal{W}$ a subset of D_G such that $\nu(X) \subseteq E_G$ if X is of sort **e** and $\nu(X) \subseteq V_G$ if it is of sort **v**. Such an assignment will be written as $\nu = (\nu_1, \ldots, \nu_n)$, with $\nu_i = \nu(X_i)$ in the usual case, where $\mathcal{W}$ is $\{X_1, \ldots, X_n\}$.

For each k-graph G, we let $\mathit{sat}(G): \Phi_k \to \mathcal{P}(\mathcal{P}(D_G)^n)$ be the mapping such that for every $\varphi(X_1, \ldots, X_n)$ in Φ_k, $\mathit{sat}(G)(\varphi)$, also denoted by $\mathit{sat}(G, \varphi)$, is the set of assignments $\nu = (\nu_1, \ldots, \nu_n) \in \mathcal{P}(D_G)^n$ such that $(G, \nu) \models \varphi$. This notation means that φ holds in G for ν. If no assignment satisfies φ in G, then $\mathit{sat}(G, \varphi) = \emptyset$.

Our first result is that the mapping sat can be computed inductively with respect to the sets of operations $\{\oplus_{n,m} \mid n, m \geqslant 0\}$, $\{\theta_{i,j,n} \mid 1 \leqslant i < j \leqslant n\}$ and $\{\sigma_{\alpha,n,p} \mid \alpha: [n] \to [p],\ n, p \geqslant 0\}$, recalled in Section 1. This shows that, given a graph G resulting from the composition of some other graphs by these operations, the set of assignments in G satisfying a CMS-formula can be computed from those in the composing graphs satisfying some CMS-formulas. We state it as follows.

Theorem 2.3. *For every k, for every φ in Φ_k, the mapping $\mathit{sat}(\varphi): \mathbf{G}_k(A) \to \mathcal{P}_f(\mathcal{P}_f(D)^n)$ such that $\mathit{sat}(\varphi)(G) = \mathit{sat}(G, \varphi)$ is H_A-inductively computable.*

Proof. Lemmas (2.4)–(2.6) prove that the family of evaluations $\{\mathit{sat}(\varphi) \mid \varphi \in \Phi_k, k \geqslant 0\}$ is H-inductive. If φ is in Φ_k, then the type of $\mathit{sat}(\varphi)$ is k. We note that Φ_k is finite up to tautological equivalence (see [14] for details). We assume that any formula in Φ_k is replaced by a minimal tautologically equivalent one (minimal with respect to some fixed lexicographical ordering that need not be specified in detail here). Hence, Φ_k is finite as well as closed under Boolean operations, and this yields the theorem. □

We need some notations and lemmas from [14]. We let $\mathcal{W} = \{X_1, \ldots, X_n\}$. If $\nu' = (\nu'_1, \ldots, \nu'_n)$ and $\nu'' = (\nu''_1, \ldots, \nu''_n)$ are two assignments in G' and G'', respectively, then the assignment $\nu := \nu' \cup \nu''$ in $G' \oplus G''$ is defined as $(\nu'_1 \cup \nu''_1, \ldots, \nu'_n \cup \nu''_n)$, where

$v'_i \cup v''_i$ is a shorthand writing of $v'(X_i) \cup v''(X_i)$ for X_i in $\mathcal{W}$. We shall keep in mind that the sets we handle are sets of n-tuples of sets.

Let us define some operations on sets of n-tuples. Two sets A and $B \subseteq \mathcal{P}_f(D)^n$ are called *separated* if

$$\left(\bigcup \{\alpha_i \mid i = 1, \ldots, n, (\alpha_1, \ldots, \alpha_n) \in A\} \right)$$

$$\cap \left(\bigcup \{\beta_i \mid i = 1, \ldots, n, (\beta_1, \ldots, \beta_n) \in B\} \right) = \emptyset.$$

If A and B are two sets of n-tuples of sets, then we define an *extended* union $\cup$ by $A \cup B = \{\alpha \cup \beta \mid \alpha \in A, \beta \in B\}$. If, in addition, A and B are *separated*, then we shall write $A \veebar B$ instead of $A \cup B$.

It is clear that $\mathbf{\emptyset} = \{(\emptyset, \emptyset, \ldots, \emptyset)\}$ is the unit of $\cup$ (and of $\veebar$), i.e. for each A, $A \cup \mathbf{\emptyset} = A$. We observe that the empty set $\emptyset$ is the zero element of $\cup$: for each A, $A \cup \emptyset = \emptyset$. (Note that $\mathbf{\emptyset}$ is an element of $\mathcal{P}_f(D)^n$ whereas $\emptyset$ denotes the usual empty set.)

The disjoint set union will be written as $\uplus$. That is, if A and B are two sets such that $A \cap B = \emptyset$, then $A \uplus B$ is nothing but $A \cup B$. We assume that $A \uplus B$ is undefined if $A \cap B \neq \emptyset$. It is evident that the unit of $\uplus$ is $\emptyset$.

Let us assume that $A \veebar B$ is not defined if A and B are not separated. Note, in particular, that $A \uplus B$ is defined if $A = \emptyset$, and that $A \veebar B$ is defined if $A = \emptyset$ or $A = \mathbf{\emptyset}$.

In Lemma 2.4, we will consider the case where a graph is obtained as the disjoint sum of two other graphs.

Lemma 2.4. *Let $k = k' + k''$. Given φ in Φ_k, one can construct a finite sequence of formulas $\psi'_1, \psi'_2, \ldots, \psi'_m$ in $\Phi_{k'}$ and a finite sequence of formulas $\psi''_1, \psi''_2, \ldots, \psi''_m$ in $\Phi_{k''}$ such that for every k'-graph G', for every k''-graph G'',*

$$\mathbf{sat}(G' \oplus G'', \varphi) = \biguplus_{1 \leqslant j \leqslant m} \mathbf{sat}(G', \psi'_j) \veebar \mathbf{sat}(G'', \psi''_j) .$$

Proof. It was proved by Courcelle [14] that, given a formula φ in Φ_k, one can construct a finite sequence of formulas $\varphi'_1, \varphi'_2, \ldots, \varphi'_{n'}$ in $\Phi_{k'}$, a finite sequence of formulas $\varphi''_1, \varphi''_2, \ldots, \varphi''_{n''}$ in $\Phi_{k''}$ and an $(n' + n'')$-place Boolean expression B such that, for every k'-graph G', for every k''-graph G'', for every assignment v' in G', for every assignment v'' in G'', if $v = v' \cup v''$ and $G = G' \oplus G''$, then

$$\varphi_G(v) = B[\varphi'_{1G'}(v'), \ldots, \varphi'_{n'G'}(v'), \varphi''_{1G''}(v''), \ldots, \varphi''_{n''G''}(v'')],$$

where $\varphi_G(v)$ denotes the Boolean value *true* iff $(G, v) \models \varphi$ and *false* otherwise; hence, $\varphi_G(v) = \textit{true}$ iff $\mathbf{sat}(G, \varphi) \neq \emptyset$.

This Boolean expression can be put in the form $\bigvee_{1 \leqslant i \leqslant r} \lambda'_{iG'}(v') \wedge \lambda''_{iG''}(v'')$, where λ'_i and λ''_i are some Boolean combinations of the φ'_j's and the φ''_j's respectively. The

formulas λ'_i and λ''_i are, respectively, in $\Phi_{k'}$ and $\Phi_{k''}$, because each set Φ_k is closed under Boolean operations. We then have the following identity:

$$\textit{sat}(G, \varphi) = \bigcup_{1 \leqslant i \leqslant r} \textit{sat}(G', \lambda'_i) \circledcirc \textit{sat}(G'', \lambda''_i). \tag{1}$$

It is easy to arrive at this statement by proving a double inclusion, i.e. that an assignment of any side of (1) belongs to the other side. (Since G' and G'' are disjoint, $\textit{sat}(G', \lambda'_i)$ and $\textit{sat}(G'', \lambda''_i)$ are separated.)

Now, we shall transform this union into a disjoint one. For each nonempty subset I of $[r]$, we let $\lambda'_I = \bigwedge_{i \in I} \lambda'_i \wedge \bigwedge_{i \notin I} \neg \lambda'_i$ and $\lambda''_I = \bigwedge_{i \in I} \lambda''_i \wedge \bigwedge_{i \notin I} \neg \lambda''_i$. We assert that

$$\textit{sat}(G, \varphi) = \biguplus_{I, J \subseteq [r],\ I \cap J \neq \emptyset} \textit{sat}(G', \lambda'_I) \circledcirc \textit{sat}(G'', \lambda''_J). \tag{2}$$

We will prove Eq. (2) by proving the two inclusions.

First, let us show that the left-hand side is included in the right-hand one. If v is an assignment belonging to $\textit{sat}(G, \varphi)$, then $v = v' \cup v''$ in a unique way. By identity (1), there exists an integer i_0 in $[r]$ such that $v' \in \textit{sat}(G', \lambda'_{i_0})$ and $v'' \in \textit{sat}(G'', \lambda''_{i_0})$.

It suffices to take $I = \{i \in [r] \mid \lambda'_{iG'}(v') = \textit{true}\}$ and $J = \{i \in [r] \mid \lambda''_{iG''}(v'') = \textit{true}\}$. Since i_0 belongs to both I and J, $I \cap J \neq \emptyset$. Therefore, the left-hand side is included in the right-hand one.

Second, we shall prove the inclusion in the other direction. To do so, consider an element v of the right-hand side; there exist two assignments v' and v'' such that $v = v' \cup v''$, $v' \in \textit{sat}(G', \lambda'_I)$, $v'' \in \textit{sat}(G'', \lambda''_J)$, where I and J are subsets of $[r]$ such that $I \cap J \neq \emptyset$. Then, we can find an integer j in $I \cap J$ such that $\lambda'_{jG'}(v')$ and $\lambda''_{jG''}(v'')$ are true, by definition of λ'_I and λ''_J. Hence, v' belongs to $\textit{sat}(G', \lambda'_j)$ and v'' belongs to $\textit{sat}(G'', \lambda''_j)$. Finally, v belongs to $\textit{sat}(G, \varphi)$.

It remains to show that the union $\uplus$ of Eq. (2) is actually disjoint. Otherwise, there exists v in $\textit{sat}(G, \varphi)$ that can be written as $v = v' \cup v''$, where $v' \in \textit{sat}(G', \lambda'_I) \cap \textit{sat}(G', \lambda'_{I_1})$ and $v'' \in \textit{sat}(G'', \lambda''_J) \cap \textit{sat}(G'', \lambda''_{J_1})$ such that $I \cap J \neq \emptyset$ and $I_1 \cap J_1 \neq \emptyset$. If the sets I and I_1 are distinct, then, for an integer i that belongs to one of these sets and not to the other, we will have λ'_i and $\neg \lambda'_i$ at the same time, which is impossible. Hence, we must have $I = I_1$ and, similarly, $J = J_1$.

By renumbering the formulas in (2), one gets an equality as stated in the lemma. □

In some applications, formula (1) may suffice. That is, one need not always take a disjoint union as in the statement (see Example 2.11).

The lemma below expresses that $\textit{sat}(\varphi)$ has a decomposition with respect to the operation $\theta_{i,j}$. Let the graph $G' = \theta_{i,j}(G)$ be the result of the fusion of the two vertices $src_G(i)$ and $src_G(j)$. Formally, this operation is defined by a surjective mapping $f: V_G \to V_{G'}$, where f maps $src_G(j)$ to $src_G(i)$, and v to v for $v \in V_{G'}$. Then for every assignment v in G, we define the assignment $v' = \theta_{i,j}(v)$ by

$v'(U) = v(U)$ for U of sort **e** (since $E_{G'} = E_G$),

$v'(U) = f(v(U))$ for U of sort **v**.

The same definition can be written in terms of tuples. That is, if $v=(v_1, \ldots, v_n)$ is an assignment in G, then $\theta_{i,j}(v)$ is the assignment defined on $\theta_{i,j}(G)$ by $\theta_{i,j}(v)=(\theta_{i,j}(v_1), \ldots, \theta_{i,j}(v_n))$.

Lemma 2.5. *Given a formula φ in Φ_k and i,j in $[k]$, $i<j$, one can construct a finite sequence of formulas $\psi_1, \ldots, \psi_m$ in Φ_k such that, for every k-graph G,*

$$\mathbf{sat}(\theta_{i,j}(G), \varphi)= \biguplus_{1 \leqslant i' \leqslant m} \mathscr{V}_{i,i'} \circledcirc \mathbf{sat}(G, \psi_{i'}),$$

where each $\mathscr{V}_{i,i'}$ is a singleton consisting of an n-tuple, the elements of which are either $\{src_G(i)\}$ or $\emptyset$.

Proof. It was proved by Courcelle [14] that, given a formula φ in Φ_k and i,j in $[k]$, $i<j$, one can construct a formula ψ in Φ_k such that for every k-graph G, for every assignment v in G, if $G'=\theta_{i,j}(G)$ and $v'=\theta_{i,j}(v)$, then $\varphi_{G'}(v')=\psi_G(v)$.

Consider first the case of a formula φ having one free variable X. If this variable is of sort **e**, then $\mathbf{sat}(G', \varphi)=\mathbf{sat}(G, \psi)$. If it is of sort **v**, then $\mathbf{sat}(G', \varphi)$ will be computed as follows.

Let $\rho_{i,j}(X)$ be the formula $\mathbf{s}_i=\mathbf{s}_j \vee (\mathbf{s}_i \notin X \wedge \mathbf{s}_j \notin X)$ and $\rho'_{i,j}(Y)$ be the formula $\mathbf{s}_i \neq \mathbf{s}_j \wedge \mathbf{s}_i \notin Y \wedge \mathbf{s}_j \notin Y$. The former says that the sources $src_G(i)$ and $src_G(j)$ are equal or both not in X, whereas the latter expresses that they are neither equal nor in Y. Note that $\rho_{i,j}(X)$ is not the negation of $\rho'_{i,j}(X)$.

We have $\mathbf{sat}(G', \varphi)=\theta_{i,j}(\mathbf{sat}(G, \psi))$. The formula ψ can be written as $(\psi \wedge \rho_{i,j}) \vee (\psi \wedge \neg \rho_{i,j})$. In other words, two cases are possible. In the first case, $\rho_{i,j}$ holds, which implies that X remains unchanged under the operation $\theta_{i,j}$, i.e. $f(X)=X$. Hence, $\theta_{i,j}(\mathbf{sat}(G, \psi \wedge \rho_{i,j}))$ is $\mathbf{sat}(G, \psi \wedge \rho_{i,j})$. Consequently, $\mathbf{sat}(G', \varphi)=\mathbf{sat}(G, \psi \wedge \rho_{i,j}) \uplus \theta_{i,j}(\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j}))$. It is a disjoint union because the two formulas cannot hold simultaneously.

Now, it remains to calculate $\theta_{i,j}(\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j}))$. In this case, at least $src_G(i)$ or $src_G(j)$ belongs to X. Let Y be a new variable and $\psi'(Y)$ the formula $\psi[S_{\{i\}}(Y)] \vee \psi[S_{\{j\}}(Y)] \vee \psi[S_{\{i,j\}}(Y)]$. We use here a new notation. For $I \subseteq \mathbb{N}$, we let $S_I(Y)$ be a new term denoting in a graph G of interest the set $Y \cup \{src_G(i) \mid i \in I\}$. $\psi[S_I(Y)]$ denotes the result of the substitution of $S_I(Y)$ for X, after some renamings have been done to the variables of ψ, as usual. The special symbols S_I can be easily eliminated: $x \in S_I(U)$ will be replaced by $x \in U \wedge \bigvee_{i \in I}(x=\mathbf{s}_i)$. Using them is just a tool for denoting complicated formulas.

We claim that

$$\theta_{i,j}(\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j}))=\{(\{src_G(i)\})\} \circledcirc \mathbf{sat}(G, \psi' \wedge \rho'_{i,j}).$$

We prove this claim in two parts.

(i) Let $U \in \theta_{i,j}(\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j}))$. Then, one can find a set U' in $\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j})$ such that $U=f(U')$. Since the mapping f transforms the sources $src_G(i)$ and $src_G(j)$

into $src_G(i)$, $f(U')$ is of the form $\{src_G(i)\} \cup U''$, where U'' is the set $U' - \{src_G(i), src_G(j)\}$. Thus, three cases are possible: $src_G(i) \in U'$; $src_G(j) \in U'$; and $src_G(i)$ and $src_G(j)$ are both in U'. It follows that U'' belongs to $\mathbf{sat}(G, \psi' \wedge \rho'_{i,j})$.

(ii) Conversely, let U'' belong to $\mathbf{sat}(G, \psi' \wedge \rho'_{i,j})$. If U' is one of the sets $U'' \cup \{src_G(i)\}$, $U'' \cup \{src_G(j)\}$ or $U'' \cup \{src_G(i), src_G(j)\}$, then it does satisfy $\psi \wedge \neg \rho_{i,j}$. Hence, $U = f(U') = U'' \cup \{src_G(i)\}$ belongs to $\theta_{i,j}(\mathbf{sat}(G, \psi \wedge \neg \rho_{i,j}))$. This completes the proof of the claim.

Finally, by collecting all these results, we have

$$\mathbf{sat}(G', \varphi) = \mathbf{sat}(G, \psi \wedge \rho_{i,j}) \uplus (\{(\{src_G(i)\})\} \circledcirc \mathbf{sat}(G, \psi' \wedge \rho'_{i,j})).$$

Or, more concisely, we can write this equality as follows:

$$\mathbf{sat}(G', \varphi) = \biguplus_{1 \leqslant k \leqslant 2} \mathscr{V}_{i,k} \circledcirc \mathbf{sat}(G, \lambda_k),$$

with $\mathscr{V}_{i,2} = \{(\emptyset)\}$, $\mathscr{V}_{i,1} = \{(\{src_G(i)\})\}$, $\lambda_2 = \psi \wedge \rho_{i,j}$ and $\lambda_1 = \psi' \wedge \rho'_{i,j}$. We obtain, then, the desired equality, as stated in Lemma 2.5.

The generalization to formulas with more than one variable is straightforward. Consider, for example, the case where a formula φ has two variables X_1 and X_2. As far as $\rho_{i,j}$ and $\rho'_{i,j}$ are concerned, they shall be extended to two variables. That is, $\rho_{i,j}(X_1, X_2)$ will be the formula $\mathbf{s}_i = \mathbf{s}_j \vee (\mathbf{s}_i \notin X_1 \wedge \mathbf{s}_j \notin X_1 \wedge \mathbf{s}_i \notin X_2 \wedge \mathbf{s}_j \notin X_2)$ and $\rho'_{i,j}(X_1, X_2)$ the formula $\mathbf{s}_i \neq \mathbf{s}_j \wedge \mathbf{s}_i \notin X_1 \wedge \mathbf{s}_j \notin X_1 \wedge \mathbf{s}_i \notin X_2 \wedge \mathbf{s}_j \notin X_2$. Similarly, if $\rho_{i,j}$ is true, then $\theta_{i,j}$ does not affect X_1 and X_2. We let ψ_{00} be the formula $\psi \wedge \rho_{i,j}$. When $\rho_{i,j}$ is false, then $src_G(i)$ or $src_G(j)$ belongs to X_1 or to X_2 or to both. This can be written, respectively, by the following formulas:

(1) $\psi_{10}(Y_1, X_2) = (\psi(S_{\{i\}}(Y_1), X_2) \vee \psi(S_{\{j\}}(Y_1), X_2) \vee \psi(S_{\{i,j\}}(Y_1), X_2))$

$$\wedge \rho'_{i,j}(Y_1, X_2),$$

(2) $\psi_{01}(X_1, Y_2) = (\psi(X_1, S_{\{i\}}(Y_2)) \vee \psi(X_1, S_{\{j\}}(Y_2)) \vee \psi(X_1, S_{\{i,j\}}(Y_2)))$

$$\wedge \rho'_{i,j}(X_1, Y_2),$$

(3) $\psi_{11}(Y_1, Y_2) = \bigvee\!\!\bigvee_{\substack{I_1, I_2 \subseteq \{i,j\}, \\ I_1, I_2 \neq \emptyset}} \psi(S_{I_1}(Y_1), S_{I_2}(Y_2)) \wedge \rho'_{i,j}(Y_1, Y_2),$

where Y_1 and Y_2 are new variables. To simplify our writing, we define sr_α by

$sr_\alpha = \{src_G(i)\}$ if $\alpha = 1$ and

$sr_\alpha = \emptyset$ if $\alpha = 0$.

Then, we have

$$\mathbf{sat}(G', \varphi) = \biguplus_{\alpha_1, \alpha_2 \in \{0,1\}} \{(sr_{\alpha_1}, sr_{\alpha_2})\} \circledcirc \mathbf{sat}(G, \psi_{\alpha_1 \alpha_2}).$$

The subscripts of these formulas are words $\alpha_1\alpha_2$ such that α_1 (α_2) is 0 if the variable X_1 (X_2) contains neither $src_G(i)$ nor $src_G(j)$ and 1 otherwise. More generally, if the formula has n free variables, then the subscript will be in $\{0, 1\}^n$ and, consequently, there will be 2^n formulas. The proof of the lemma is achieved. □

Lemma 2.6 deals with the source redefinition operation.

Lemma 2.6. *Given φ in Φ_k, and $\alpha:[k]\to[p]$, one can construct a formula φ' in Φ_p such that, for every p-graph G,*

$$\textit{sat}(\sigma_\alpha(G), \varphi) = \textit{sat}(G, \varphi').$$

Proof. This is just another form of Lemma 4.7 of [14]. □

The purpose of the following definition is to unify the results of Lemmas 2.4–2.6 and to extend them to derived operations.

Definition 2.7. A $(\uplus, \Cup)$-*polynomial* is an expression of the form $\emptyset$, a monomial or a sum of the form $m_1 \uplus m_2 \uplus \cdots \uplus m_k$, where each term $m_1, \ldots, m_k$ is a monomial. A *monomial* is an expression of the form $\emptyset$, t_1 or $t_1 \Cup t_2 \Cup \cdots \Cup t_n$, where each t_i is either a *variable* or a *constant*, denoting a fixed element of $\mathscr{P}_f(\mathscr{P}_f(D)^n)$. Similarly, one gets the notion of a $(\cup, \cup)$-polynomial, a $(\cup, \Cup)$-polynomial and a $(\uplus, \cup)$-polynomial.

The decomposition operators constructed in Lemmas 2.4–2.6 are $(\cup, \Cup)$-polynomials.

In the structure $\mathscr{D} = \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \cup, \emptyset, \emptyset \rangle$, a $(\cup, \cup)$-polynomial p denotes a total function $\mathscr{P}_f(\mathscr{P}_f(D)^n)^m \to \mathscr{P}_f(\mathscr{P}_f(D)^n)$, where m is the number of variables of p. A polynomial of the three other types denotes a partial function since $\uplus$ and $\Cup$ are partial operations.

A *semiring* is a structure $\mathscr{R} = \langle S, \sqcup, \sqcup\!\!\sqcup, \bot, \varepsilon \rangle$, where

- S is a nonempty set,
- $\sqcup$ and $\sqcup\!\!\sqcup$ are two binary total operations on S,
- $(S, \sqcup, \bot)$ is a commutative monoid,
- $(S, \sqcup\!\!\sqcup, \varepsilon)$ is a monoid with zero element $\bot$ (i.e. $a \sqcup\!\!\sqcup \bot = \bot \sqcup\!\!\sqcup a = \bot$ for all a in S) and
- $\sqcup\!\!\sqcup$ distributes over $\sqcup$, that is,

$$a \sqcup\!\!\sqcup (b \sqcup c) = (a \sqcup\!\!\sqcup b) \sqcup (a \sqcup\!\!\sqcup c)$$

and

$$(b \sqcup c) \sqcup\!\!\sqcup a = (b \sqcup\!\!\sqcup a) \sqcup (c \sqcup\!\!\sqcup a).$$

These laws can be put in the form of equations and they make it possible to transform every derived operation written with $\sqcup$, $\sqcup\!\!\sqcup$ and constants into a $(\sqcup, \sqcup\!\!\sqcup)$-polynomial denoting the same function. The structure $\langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \cup, \emptyset, \emptyset \rangle$ is a semiring.

Since $\uplus$ and $\biguplus$ are partial operations, the case of functions written with them is not so straightforward. The structure $\mathscr{D} = \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \uplus, \biguplus, \emptyset, \boldsymbol{\emptyset} \rangle$ satisfies the following identities:

(s_1) $x \uplus y \simeq y \uplus x$,

(s_2) $x \uplus (y \uplus z) \simeq (x \uplus y) \uplus z$,

(s_3) $x \uplus \emptyset = \emptyset \uplus x = x$,

(s_4) $x \biguplus (y \biguplus z) \simeq (x \biguplus y) \biguplus z$,

(s_5) $x \biguplus \boldsymbol{\emptyset} = \boldsymbol{\emptyset} \biguplus x = \boldsymbol{\emptyset}$,

(s_6) $x \biguplus (y \uplus z) \simeq (x \biguplus y) \uplus (x \biguplus z)$,

(s_7) $(x \uplus y) \biguplus z \simeq (x \biguplus z) \uplus (y \biguplus z)$.

In these identities, $=$ means that both sides are always defined and equal, whereas $\simeq$ means that whenever one side is defined, so is the other and they are equal. We shall use them as rewriting rules in both directions for (s_1), (s_2) and (s_4), and from left to right for the others.

In this way, every term $t(x_1, \ldots, x_n)$ over $\uplus$, $\biguplus$ can be transformed into a $(\uplus, \biguplus)$-polynomial $p(x_1, \ldots, x_n)$ such that, whenever $t(x_1, \ldots, x_n)$ is defined, then so is $p(x_1, \ldots, x_n)$ and their values are equal. (The example of $(x_1 \uplus x_1) \biguplus \boldsymbol{\emptyset}$ reducing to $\boldsymbol{\emptyset}$ shows that p is, in general, more defined than t.)

It follows, in particular, that the function obtained by substituting a polynomial for a variable in a polynomial can be transformed into a polynomial. For example,

$$a \uplus (x \biguplus y)[b \uplus (z \biguplus c)/x, e \uplus (t \biguplus d)/y] \to a \uplus [(b \uplus (z \biguplus c)) \biguplus (e \uplus (t \biguplus d))]$$
$$\to a \uplus (b \biguplus e) \uplus (b \biguplus t \biguplus d) \uplus (z \biguplus c \biguplus e)$$
$$\uplus (z \biguplus c \biguplus t \biguplus d).$$

Hence, we have the following corollary.

Corollary 2.8. *Let F be a derived signature of H_A. For every k and φ in Φ_k, the mapping $\boldsymbol{sat}(\varphi)$ is F-inductively computable. Moreover, the corresponding decomposition operators are $(\uplus, \biguplus)$-polynomials.*

Now, we shall define a class of evaluations on graphs that can be computed inductively, in some sense "directly".

Definition 2.9 (*MS-evaluation*). An *evaluation structure* is a structure $\mathscr{R} = \langle S, \sqcup, \dot{\sqcup}, \bot, \varepsilon \rangle$. In many cases, it will be a semiring, but we do not require this in general. The notion of a homomorphism $h: \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \dot{\cup}, \emptyset, \boldsymbol{\emptyset} \rangle \to \mathscr{R}$ is standard. (Note the correspondence of $\cup$ with $\sqcup$, $\dot{\cup}$ with $\dot{\sqcup}$, $\emptyset$ with $\bot$ and $\boldsymbol{\emptyset}$ with ε.)

The following weaker notions will be useful. A $(\uplus, \biguplus)$-*homomorphism* $h: \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \uplus, \biguplus, \emptyset, \boldsymbol{\emptyset} \rangle$ into $\mathscr{R}$ is a mapping such that

$$h(\emptyset) = \bot,$$

$$h(\boldsymbol{\emptyset}) = \varepsilon,$$

$$h(A \uplus B) = h(A) \sqcup h(B) \quad \text{if } A \cap B = \emptyset,$$

$$h(A \uplus\!\!\uplus B) = h(A) \sqcup\!\!\sqcup h(B) \quad \text{if } A \text{ and } B \text{ are separated.}$$

Similarly, we define $(\cup, \uplus\!\!\uplus)$-homomorphisms and $(\uplus, \cup)$-homomorphisms. As we shall see later in more detail, the cardinality mapping ($\mathbf{Card}: \mathscr{P}_f(\mathscr{P}_f(D)^n) \to \mathbb{N}$) is a $(\uplus, \uplus\!\!\uplus)$-*homomorphism* but not a homomorphism.

Let us recall that for dealing with graphs, we let D be a countable set such that $D_G = V_G \cup E_G$ is a finite subset of D for every graph G. (In a concrete implementation, D is the set of memory locations that one considers, as usual, to be infinite.)

An *MS-evaluation* is a mapping $v: \mathbf{G}_k(A) \to S$ where $\mathscr{R} = \langle S, \sqcup, \sqcup\!\!\sqcup, \perp, \varepsilon \rangle$ is an evaluation structure, that is of the form $v = h \circ \boldsymbol{sat}(\varphi)$, where φ is a CMS-formula with free variables in $\{X_1, \ldots, X_n\}$, and h is a $(\uplus, \uplus\!\!\uplus)$-homomorphism $\langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \uplus, \uplus\!\!\uplus, \emptyset, \boldsymbol{\emptyset} \rangle \to \mathscr{R}$. We must remember that, as defined in Section 2, $\boldsymbol{sat}(\varphi)$ is a mapping $\mathbf{G}_k(A) \to \mathscr{P}_f(\mathscr{P}_f(D)^n)$. Hence, $h \circ \boldsymbol{sat}(\varphi)$ is a mapping $\mathbf{G}_k(A) \to S$.

In words, an MS-evaluation is defined by a homomorphism which takes as input the tuples computed by the mapping ***sat***. Thus, its value for a graph G can be actually determined from those for the subgraphs composing G. Moreover, its values for the basic graphs (the empty graph, or graphs reduced to single edges or vertices) must be known since these graphs are the leaves of the parse tree of graphs.

Theorem 2.10. *Every MS-evaluation is H_A-inductively computable.*

Proof. This is a consequence of Theorem 2.3. In fact, it suffices to compose h and ***sat*** in the three lemmas. We have, with the same notations as in Lemma 2.4,

$$h(\boldsymbol{sat}(G' \oplus G'', \varphi)) = h\left(\biguplus_{1 \leqslant j \leqslant m} \boldsymbol{sat}(G', \psi'_j) \uplus\!\!\uplus \boldsymbol{sat}(G'', \psi''_j) \right)$$

$$= \bigsqcup_{1 \leqslant j \leqslant m} h(\boldsymbol{sat}(G', \psi'_j) \uplus\!\!\uplus \boldsymbol{sat}(G'', \psi''_j))$$

$$= \bigsqcup_{1 \leqslant j \leqslant m} h(\boldsymbol{sat}(G', \psi'_j)) \sqcup\!\!\sqcup h(\boldsymbol{sat}(G'', \psi''_j)).$$

Hence,

$$v_\varphi(G' \oplus G'') = \bigsqcup_{1 \leqslant j \leqslant m} v_{\psi'_j}(G') \sqcup\!\!\sqcup v_{\psi''_j}(G''),$$

where, for $\varphi \in \Phi_k$, $v_\varphi(G)$ denotes $h(\boldsymbol{sat}(G, \varphi))$.
Similarly, we get from the other lemmas,

$$h(\boldsymbol{sat}(\theta_{i,j}(G), \varphi)) = \bigsqcup_{1 \leqslant i' \leqslant m} h(\mathscr{V}_{i,i'}) \sqcup\!\!\sqcup h(\boldsymbol{sat}(G, \psi_{i'}))$$

and

$$h(\mathbf{sat}(\sigma_\alpha(G), \varphi)) = h(\mathbf{sat}(G, \varphi')).$$

Finally,

$$v_\varphi(\theta_{i,j}(G)) = \bigsqcup_{1 \leq i' \leq m} h(\gamma_{i,i'}) \sqcup v_{\psi_{i'}}(G)$$

and

$$v_\varphi(\sigma_\alpha(G)) = v_{\varphi'}(G).$$

Hence, this proves that $\mathcal{E} = \{h \circ \mathbf{sat}(\varphi) \mid \varphi \in \Phi_k\}$ is H_A-inductive.

Example 2.11. We give a list of evaluations. Some of them are MS-evaluations, others are not. (We shall discuss in Section 3 the appropriate data structures for computing them.) We let A consist of symbols of type 2. Hence, we consider directed graphs, and not hypergraphs. We also let $k=2$ and $\varphi(X)$ be the MS-formula stating that X is the set of edges of a simple path (i.e. a path without cycles) linking the first source to the second one. (Such a formula has been constructed in [14].) Let $G \in \mathbf{G}_2(A)$. We shall consider the following evaluations expressed in terms of $\mathbf{sat}(\varphi)(G)$, described in Fig. 1.

Evaluation	Definition	Description
$v_1(G)$	$\mathbf{sat}(\varphi)(G)$	The set of all simple paths from $src_G(1)$ to $src_G(2)$.
$v_2(G)$	$\mathbf{Card}(\mathbf{sat}(\varphi)(G))$	The number of simple paths from $src_G(1)$ to $src_G(2)$.
$v_3(G)$	$\mathbf{Nonempty}(\mathbf{sat}(\varphi)(G))$	This evaluation has the value **true** if $\mathbf{sat}(\varphi)(G) \neq \emptyset$ and **false** otherwise. It indicates the existence of a simple path from $src_G(1)$ and $src_G(2)$.
$v_4(G)$	$\mathbf{Max}\{\mathbf{Card}(X) \mid X \in \mathbf{sat}(\varphi)(G)\}$	The maximal length of a simple path from $src_G(1)$ to $src_G(2)$. (If $v_1(G)=\emptyset$, then $v_4(G)=-\infty$.)
$v_5(G)$	$\mathbf{Min}\{\mathbf{Card}(X) \mid X \in \mathbf{sat}(\varphi)(G)\}$	The length of a shortest path from $src_G(1)$ to $src_G(2)$ $(=+\infty$ if $v_1(G)=\emptyset)$.
$v_6(G)$	$\Sigma\{\mathbf{Card}(X) \mid X \in \mathbf{sat}(\varphi)(G)\}$	The sum of the lengths of all simple paths from $src_G(1)$ to $src_G(2)$.
$v_7(G)$	$\mathbf{Average}(\mathbf{sat}(\varphi)(G))$	The average length of a simple path from $src_G(1)$ to $src_G(2)$ $(=v_6(G)/v_2(G))$.
$v_8(G)$	$\mathbf{Dif}(\mathbf{sat}(\varphi)(G))$	The difference between the maximum and the minimum lengths of simple paths from $src_G(1)$ to $src_G(2)$. That is, $v_4(G)-v_5(G)$ $(=-\infty$ if $v_1(G)=\emptyset)$.

Fig. 1. Some evaluations.

MS-evaluation	S	$\sqcup$	$\sqcup\!\!\sqcup$	$\perp$	ε
v_2	$\mathbb{N}$	$+$	$\times$	0	1
v_3	{**true**, **false**}	$\vee$	$\wedge$	**false**	**true**
v_4	$\mathbb{N}\cup\{-\infty\}$	**Max**	$+$	$-\infty$	0
v_5	$\mathbb{N}\cup\{+\infty\}$	**Min**	$+$	$+\infty$	0

Fig. 2. Evaluation structures.

The evaluations v_2, v_3, v_4, v_5 are MS-evaluations. The corresponding evaluation structures are listed in Fig. 2. The others, v_6, v_7, v_8, are not, but they are computable in terms of auxiliary MS-evaluations so that they are H_A-inductively computable. The complexities of the corresponding algorithms will be discussed in Section 3. Let us check that v_2 is really an MS-evaluation. If A and B are two elements of $\mathscr{P}_f(\mathscr{P}_f(D))$ (two finite sets of finite subsets of D), then

$$\mathbf{Card}(A \uplus B) = \mathbf{Card}(A) + \mathbf{Card}(B).$$

We also have

$$\mathbf{Card}(A \circledcirc B) = \mathbf{Card}(\{\alpha\cup\beta \mid \alpha\in A,\ \beta\in B\})$$

$$= \mathbf{Card}(A) \times \mathbf{Card}(B)$$

because A and B are separated, i.e. are such that $(\bigcup\{\alpha_i \mid i=1,\ldots,n,\ (\alpha_1,\ldots,\alpha_n)\in A\})\cap(\bigcup\beta_i \mid i=1,\ldots,n,\ (\beta_1,\ldots,\beta_n)\in B\})=\emptyset$. Finally,

$$\mathbf{Card}(\emptyset) = 0,$$

$$\mathbf{Card}(\boldsymbol{\emptyset}) = 1.$$

Note the difference between $\boldsymbol{\emptyset}$ and $\emptyset$ in the current example. In fact, $\boldsymbol{sat}(\varphi)(G)=\emptyset$ means that there is no path between the first and the second source, whereas $\boldsymbol{\emptyset}\in\boldsymbol{sat}(\varphi)(G)$ means that the empty path links them (i.e. they are equal).

Hence, **Card** is a $(\uplus, \circledcirc)$-homomorphism of $\langle \mathscr{P}_f(\mathscr{P}_f(D)), \uplus, \circledcirc, \emptyset, \boldsymbol{\emptyset}\rangle$ into $\langle \mathbb{N}, +, \times, 0, 1\rangle$ and v_2 is an MS-evaluation. Similarly, we can prove that v_4 and v_5 are MS-evaluations. Other interesting evaluation structures will be presented in Section 3.

3. Building algorithms

We now explain how algorithms can be constructed to compute inductive evaluations on graphs given by terms defining them. We first specify carefully the problem.

Let F be a finite derived signature of H_A (its set of sorts is a finite subset $\mathscr{S}(F)$ of $\mathbb{N}$). Let $k\in\mathscr{S}(F)$ and v be an evaluation $\mathbf{G}_k(A)\to S$ for some set S.

3.1. The basic algorithm

A (v, k, F)-*algorithm* is an algorithm that takes as input a term t in $M(F)_k$, denoting a graph G in $\mathbf{G}_k(A)$, and produces the value $v(G)$. Letting the **size** of a graph G be $\mathbf{size}(G)=\mathbf{Card}(V_G)+\mathbf{Card}(E_G)$, we have $\mathbf{size}(G)\leqslant m\cdot|t|$ for some constant m, where $t\in M(F)$ and $G=\mathbf{val}(t)$. Conversely, $|t|\leqslant m'\cdot\mathbf{size}(G)+1$ for some constant m'. More precisely, every term t denoting G can be reduced into one, t', denoting G, such that $|t'|\leqslant m'\cdot\mathbf{size}(G)+1$ by deleting some redundant parts of t. We do not wish to detail this small technical point. It follows that if an algorithm decides a graph property in time $O(|t|^k)$, where t denotes G, then one can also say that its time complexity is $O(\mathbf{size}(G)^k)$.

Proposition 3.1. *If v is F-inductively computable, then there exists a (v, k, F)-algorithm with time complexity $O(|t|\cdot\eta)$, where η is an upper bound on the complexity of the computation of each right-hand side of an equation, as in Definition* 1.1.

Proof. We first present a basic algorithm and we shall describe later how to improve it.

Let $(\mathscr{E}_s)_{s\in\mathscr{S}(F)}$ be the finite set of evaluations that we have by the definition of an inductively computable evaluation.

Let $t\in M(F)_k$ denote G and be considered, as usual, as a finite tree. Each node u of t has a label f in F. The sort of f will be called the *sort of* u, and is actually the common type of the graphs defined by the term t/u, namely the subtree issued from node u of t.

With each node u of t, we associate *attribute* occurrences $w(u)$, for each $w\in\mathscr{E}_s$, where s is the sort of u. The intended valuc of an attribute occurrence $w(u)$ is $w(G_u)$, where G_u is denoted by t/u. We now explain how it can be computed from the values of other attributes at the successor nodes of u.

Let u be a node, $w(u)$ an attribute at u, and f the symbol of F that labels u. Then, by Definition 1.1,

$$w(u)=\theta_{w,f}(w_{1,1}(u_1),\ldots,w_{1,m_1}(u_1),w_{2,1}(u_2),\ldots,w_{2,m_2}(u_2),\ldots,w_{n,m_n}(u_n)), \tag{1}$$

where $(\theta_{w,f}, w_{1,1},\ldots,w_{n,m_n})$ is the decomposition of w relative to f, and $u_1,\ldots,u_n$ is the sequence of successors of u. (This assumes that the rank of f is n; if $n=0$, then $\theta_{w,f}$ is a constant value.)

It is, thus, clear that one can compute bottom-up on the tree t all the attributes associated with all its nodes. We are actually in the case of a purely synthesized attribute grammar. See [17] for a survey of attribute grammars and their evaluation algorithms. Among the attributes of the root, one finds $v(\varepsilon)=v(G)$, namely the value to be returned as output of the algorithm.

The time complexity can be evaluated as

$$\sum_{u\in\mathbf{Node}(t)}\ \sum_{w\in\mathscr{E}_{\sigma(u)}}\eta(w,u), \tag{2}$$

where $\eta(w, u)$ is the time complexity for the evaluation of $w(u)$ by Eq. (1). An upper bound can be given as follows:

$$|t| \cdot \mathbf{Max}\{\mathbf{Card}(\mathscr{E}_s) | s \in \mathscr{S}(F)\} \cdot \eta, \tag{3}$$

where $\eta(w, u) \leqslant \eta$ for all w and u. □

3.2. An improved algorithm

Rather than computing all attributes at all nodes of t, one can compute only those that are useful for the final result, namely for $v(\varepsilon)$.

To do so, one can use a preliminary top-down pass on the tree that determines the necessary attributes. The only necessary attribute at the root is $v(\varepsilon)$. Assuming that W is the set of necessary attributes at a node u labeled by f, then the necessary attributes at u_i (the ith successor of u) are those of the form $w'(u_i)$, where $w' = w_{i,j}$ and $(\theta_{w,f}, w_{1,1}, \ldots, w_{i,j}, \ldots)$ is the decomposition of some w in W relative to f.

It is not possible to decide at this abstract level of presentation when this optimization is actually interesting. Our subsequent considerations relative to complexity will be based on formula (2) and its approximation (3). Any improvement obtained from them, say by limiting the sizes of the sets $\mathscr{E}_s$ or the values $\eta(w, u)$, will apply both to the basic algorithm and to its improved version.

Remark. For $A \subseteq \mathscr{P}_f(D)^n$, we let $\mu(A)$ be the least set of subsets of $\mathscr{P}_f(D)^n$ such that

(i) $A \in \mu(A)$,

(ii) if $B = C \uplus D$ and $B \in \mu(A)$, then $C, D \in \mu(A)$ and

(iii) if $B = C \circledcirc D$ and $B \in \mu(A)$, then $C, D \in \mu(A)$.

It is clear that, when we use the improved algorithm for evaluating $\boldsymbol{sat}(G, \varphi)$ for some formula φ, the auxiliary sets $\boldsymbol{sat}(G', \psi)$ that are needed are all in $\mu(\boldsymbol{sat}(G, \varphi))$.

3.3. Issues for the construction of efficient (v, k, F)-algorithms

The usability of this technique depends on the following facts:

(1) For each w, f, one needs to define a subroutine implementing $\theta_{w,f}$. Clearly, good data structures for storing and computing the values of attributes must be designed.

(2) It is clear that the use of as few sorts as possible, and as small sets $\mathscr{E}_s$ as possible, improves time and space complexity.

We first comment on fact (2), and propose two methods that help reduce the number of sorts and the sizes of sets $\mathscr{E}_s$. The basic ideas are to use derived operations and to avoid logic.

3.4. Using derived operations

We first consider the example of the set L of "two-terminal" directed series–parallel graphs. We let e denote the graph of the form $1 \bullet \longrightarrow \bullet 2$, with two sources linked by

a directed edge from the first source to the second one. We let $\|$ be the *parallel composition* of 2-graphs, defined by

$$G \| G' = \sigma_\alpha(\theta_{1,3}(\theta_{2,4}(G \oplus G'))),$$

where $\alpha(1)=1$ and $\alpha(2)=2$. This operation glues G and G' by fusing their ith sources, for $i=1, 2$. We let $\bullet$ be the *series composition* of 2-graphs, defined by

$$G \bullet G' = \sigma_\beta(\theta_{2,3}(G \oplus G')),$$

where $\beta(1)=1$ and $\beta(2)=4$. This operation glues G and G' by fusing the second source of G with the first source of G', and keeping as new sources the first of G and the second of G'.

Then L is the least subset of $\mathbf{G}_2(A)$ containing e and closed by $\|$ and $\bullet$. Hence, every graph in L can be expressed in terms of two operations (with a single sort, namely 2) and one constant. For expressing them in terms of the basic operations, one would need to use the following operations:

$\oplus$ of profile $2 \times 2 \to 4$,

$\theta_{1,3}, \theta_{2,3}, \theta_{2,4}$ of profile $4 \to 4$,

$\sigma_\alpha, \sigma_\beta$ of profile $4 \to 2$ and

the constant e of sort 2.

Hence, we would use six operations instead of two, and two sorts instead of one. The improvement is fairly clear.

Another example can be found in [2]. Derived operations of sorts $0, \ldots, k$ are used to generate the graphs of tree-width at most k, whereas the use of the basic operations would need the use of sorts up to $2k$.

Note that when choosing a set F of operations generating a set of graphs of interest K, one should also consider the existence of a *parsing* algorithm that, given $G \in K$, produces as efficiently as possible a term $t \in M(F)$ defining G. One should do this because the (v, k, F)-algorithm applies to a term t defining the graph of interest G. The construction of t is linear in the cases of series–parallel graphs (see [27]) and polynomial in that of partial k-trees (for fixed $k \geqslant 3$; see [1, 4]). Efficient algorithms have been given by Lagergren [23], Bodlaender and Kloks [11], and Hohberg and Reischuk [21].

3.5. *Avoiding logic*

Theorems 2.3 and 2.10 are stated in terms of logical formulas. However, it seems intractable to use them in the way they are established, because the proofs involve very large sets of auxiliary formulas.

In concrete cases, one should rather work in terms of graph properties, knowing what they mean, and forgetting the logical formulas. One can enrich the logic with auxiliary predicates, that do not increase the power but shorten the writings, as done in [12].

We illustrate this with an example, namely the construction of all *3-vertex colorings* of series–parallel graphs. Series–parallel graphs will, of course, be expressed in terms of $\|$, $\bullet$ and e, as explained above. There exists a monadic second-order formula $\varphi(X_1, X_2)$ expressing that, in a given graph, X_1, X_2 and $V_G - (X_1 \cup X_2)$ are sets of vertices defining a 3-vertex coloring of G (where $x \in X_i$ iff x has color i for $i = 1, 2$, and $x \in V_G - (X_1 \cup X_2)$ iff x has color 3).

Hence, we wish to express

$$\textbf{3-col}(G) = \textbf{\textit{sat}}(\varphi)(G) \in \mathscr{P}_f(\mathscr{P}_f(V_G)^2)$$

as an inductively computable mapping over $\langle \mathbf{G}_2(A), \|, \bullet, e \rangle$.

We shall give two constructions. The first one uses $(\uplus, \circledcirc)$-polynomials. For every graph G of type 2, with distinct sources, we let, for $i, j \in \{1, 2, 3\}$,

$$C_{i,j}(G) = \{(X_1, X_2) \mid X_1, X_2 \subseteq V_G - \{src_G(1), src_G(2)\}, \text{ and } (X_1, X_2, X_3) \text{ defines a 3-coloring of } G, \text{ where } src_G(1) \text{ gets color } i \text{ and } src_G(2) \text{ gets color } j \text{ and every } x \in X_i \text{ gets color } i\}.$$

In this definition, we let

$$X_3 := V_G - (\{src_G(1), src_G(2)\} \cup X_1 \cup X_2).$$

We then get the following inductive definitions:

(1) If $G = e$, then

$$C_{i,i}(G) := \emptyset \quad \text{for } i = 1, 2, 3,$$

$$C_{i,j}(G) := \{(\emptyset, \emptyset)\} = \boldsymbol{\emptyset} \quad \text{for } 1 \leqslant i \neq j \leqslant 3.$$

(2) If $G = G_1 \| G_2$, then

$$C_{i,j}(G) := C_{i,j}(G_1) \circledcirc C_{i,j}(G_2).$$

(3) If $G = G_1 \bullet G_2$, then

$$C_{i,j}(G) := \biguplus_{1 \leqslant k \leqslant 3} C_{i,k}(G_1) \circledcirc s_k \circledcirc C_{k,j}(G_2),$$

where

$$s_1 := \{(\{src_{G_1}(2)\}, \emptyset)\},$$

$$s_2 := \{(\emptyset, \{src_{G_1}(2)\})\},$$

$$s_3 := \{(\emptyset, \emptyset)\}.$$

(Recall that when constructing G from G_1 and G_2, one deletes the first source of G_2, so that the "middle" vertex of G is $src_{G_1}(2)$.)

Finally, the desired set is

$$\textbf{3-col}(G) := \biguplus_{1 \leqslant i, j \leqslant 3} C_{i,j}(G) \circledcirc s'_{i,j},$$

where

$$s'_{1,1} := \{(\{src_G(1), src_G(2)\}, \emptyset)\},$$

$$s'_{1,2} := \{(\{src_G(1)\}, \{src_G(2)\})\},$$

$$s'_{1,3} := \{(\{src_G(1)\}, \emptyset)\},$$

$$s'_{2,1} := \{(\{src_G(2)\}, \{src_G(1)\})\},$$

$$s'_{2,2} := \{(\emptyset, \{src_G(1), src_G(2)\})\},$$

$$s'_{2,3} := \{(\emptyset, \{src_G(1)\})\},$$

$$s'_{3,1} := \{(\{src_G(2)\}, \emptyset)\},$$

$$s'_{3,2} := \{(\emptyset, \{src_G(2)\})\},$$

$$s'_{3,3} := \{(\emptyset, \emptyset)\}.$$

We give a second definition using $(\uplus, \cup)$-polynomials. We let

$$D_{i,j}(G) = \{(Y_1, Y_2) \mid Y_1, Y_2 \subseteq V_G, \text{ and } (Y_1, Y_2, Y_3) \text{ defines a 3-coloring of } G, \text{ where } src_G(1) \in Y_i,\ src_G(2) \in Y_j,\ Y_3 = V_G - (Y_1 \cup Y_2) \text{ and } x \text{ gets color } i \text{ iff } x \in Y_i\}.$$

We then have the following:

(1) If $G = e$, then

$$D_{i,i}(G) := \emptyset \quad \text{for } i = 1, 2, 3,$$

$$D_{1,2}(G) := \{(\{src_G(1)\}, \{src_G(2)\})\},$$

$$D_{1,3}(G) := \{(\{src_G(1)\}, \emptyset)\},$$

$$D_{2,1}(G) := \{(\{src_G(2)\}, \{src_G(1)\})\},$$

$$D_{2,3}(G) := \{(\emptyset, \{src_G(1)\})\},$$

$$D_{3,1}(G) := \{(\{src_G(2)\}, \emptyset)\},$$

$$D_{3,2}(G) := \{(\emptyset, \{src_G(2)\})\}.$$

(2) If $G = G_1 \parallel G_2$, then

$$D_{i,j}(G) := D_{i,j}(G_1) \cup D_{i,j}(G_2).$$

(3) If $G = G_1 \bullet G_2$, then

$$D_{i,j}(G) := \biguplus_{1 \leqslant k \leqslant 3} D_{i,k}(G_1) \cup D_{k,j}(G_2).$$

4. A catalogue of evaluation structures

We review the evaluation structures already presented in Example 2.11, indicate the relevant data structures, and discuss briefly the complexity. They are presented in the order of increasing complexity. We also indicate some extensions to weighted graphs and to linearly ordered ones.

4.1. Boolean values

The evaluation structure is $\mathscr{B} = \langle \{\mathbf{true}, \mathbf{false}\}, \vee, \wedge, \mathbf{false}, \mathbf{true} \rangle$. Each attribute has a Boolean value.

Let p map A to **true** iff $A \neq \emptyset$. Then p is a homomorphism of $\langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \cup, \emptyset, \emptyset \rangle$ into $\mathscr{B}$. We have $p(\mathbf{sat}(G, \varphi)) = \mathbf{true}$ iff $\mathbf{sat}(G, \varphi) \neq \emptyset$, iff φ is satisfiable in G for some assignment.

The complexity parameter $\eta(w, u)$ is bounded by a constant depending linearly on the length of the decomposition operator $\theta_{w,f}$, where f labels u. The corresponding (v, k, F)-algorithm is linear in $|t|$, where t is the input term.

4.2. Cardinality

We consider here $\mathbf{Card}: \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \cup, \emptyset, \emptyset \rangle \to \langle \mathbb{N}, +, \times, 0, 1 \rangle$. It is a $(\uplus, \circledcirc)$-homomorphism. If φ has one free variable, then $\mathbf{Card}(\mathbf{sat}(G, \varphi))$ is the number of sets X satisfying φ in G.

Integers can represent the values of the corresponding attributes. As before, $\eta(w, u)$ is bounded by a constant depending linearly on the length of $\theta_{w,f}$, where f labels u.

By using this structure, one can count, e.g., the number of Hamiltonian circuits or the number of perfect matchings in a graph. (A *perfect matching* is a set of pairwise nonadjacent edges X such that every vertex belongs to some edge in X.)

4.3. Maximum and minimum cardinalities

We let here $n = 1$, and we only consider the maximum. The function $\mathbf{MaxCard}: \mathscr{P}_f(\mathscr{P}_f(D)) \to \mathbb{N} \cup \{-\infty\}$, defined by $\mathbf{MaxCard}(A) = \mathbf{Max}(\{\mathbf{Card}(\alpha) \mid \alpha \in A\})$, is a $(\uplus, \circledcirc)$-homomorphism in $\langle \mathbb{N} \cup \{-\infty\}, \mathbf{Max}, +, -\infty, 0 \rangle$ (with $\mathbf{Max}(\emptyset) = -\infty$).

Note that here one can deal with $\cup$ instead of $\uplus$, since $\mathbf{MaxCard}(A \cup B) = \mathbf{Max}\{\mathbf{MaxCard}(A), \mathbf{MaxCard}(B)\}$ even if $A \cap B \neq \emptyset$. Actually, **MaxCard** is a $(\cup, \circledcirc)$-homomorphism. As before, η depends linearly on the sizes of the operators $\theta_{w,f}$.

With this structure and the dual one for the minimum, one can express some $\mathscr{NP}$-complete problems (numbered as in [18]) such as

- Vertex cover [GT1],
- Minimum maximal matching [GT10],
- Clique [GT19],

- Independent set [GT20],
- Induced subgraph with property π [GT21] (for a monadic second-order property π) and
- Planar subgraph [GT27].

To be more precise, we obtain that the following functions are MS-evaluations:

$$f_1(G) = \mathbf{Min}\{\mathbf{Card}(X) \mid X \subseteq V_G, \text{ every edge of } G \text{ has at least one vertex in } X\}.$$

$$f_{10}(G) = \mathbf{Min}\{\mathbf{Card}(X) \mid X \subseteq E_G, \text{ no two edges have a common vertex and every vertex of an edge not in } X \text{ belongs to some edge in } X\}.$$

$$f_{19}(G) = \mathbf{Max}\{\mathbf{Card}(X) \mid X \subseteq V_G, \text{ every two vertices of } X \text{ are adjacent}\}.$$

$$f_{20}(G) = \mathbf{Max}\{\mathbf{Card}(X) \mid X \subseteq V_G, \text{ no two vertices of } X \text{ are adjacent}\}.$$

$$f_{21}(G) = \mathbf{Max}\{\mathbf{Card}(X) \mid X \subseteq V_G, \text{ the induced subgraph of } G \text{ with a set of vertices } X \text{ satisfies } \pi\}, \text{ where } \pi \text{ is any monadic second-order graph property.}$$

$$f_{27}(G) = \mathbf{Max}\{\mathbf{Card}(X) \mid X \subseteq E_G \text{ such that } X \text{ defines a planar subgraph of } G\}.$$

For $i \in \{1, 10\}$, the problem [GTi] consists in deciding whether $f_i(G) \leqslant k$ for a given G and k. For $i \in \{19, 20, 21, 27\}$, the problem [GTi] consists in deciding whether $f_i(G) \geqslant k$ for a given G and k. One obtains linear algorithms

Bern et al. [6] show how the *irredundance number* of a tree can be computed in linear time. By our technique, we can compute this number in linear time for every graph given by a term. (A subset X of V_G is *redundant* if there is a proper subset X' of X such that $X' \cup \{\text{the set of vertices adjacent to some vertex of } X'\} = X \cup \{\text{the set of vertices adjacent to some vertex of } X\}$. It is *maximal irredundant* if it is not redundant and if every set of vertices of the form $X \cup \{v\}$, where $v \notin X$, is redundant. It is clear that an MS-formula $\varphi(X)$ can be constructed to express that X is maximal irredundant. The irredundance number of G is then the minimum cardinality of a maximal irredundant subset of V_G. This number is of the form $\mathbf{MinCard}(\boldsymbol{sat}(G, \varphi))$.)

4.4 Sum of cardinalities

We also let $n = 1$. We consider $\sum \mathbf{Card}(A) = \sum\{\mathbf{Card}(\alpha) \mid \alpha \in A\}$, so that $\sum \mathbf{Card}$ maps $\mathscr{P}_f(\mathscr{P}_f(D))$ into $\mathbb{N}$. It is clear that

$$\sum \mathbf{Card}(A \uplus B) = \sum \mathbf{Card}(A) + \sum \mathbf{Card}(B).$$

The definition of $\sum \mathbf{Card}(A \circledcirc B)$ needs the auxiliary use of $\mathbf{Card}(A)$ and $\mathbf{Card}(B)$:

$$\sum \mathbf{Card}(A \circledcirc B) = \mathbf{Card}(B) \cdot \sum \mathbf{Card}(A) + \mathbf{Card}(A) \cdot \sum \mathbf{Card}(B).$$

(The verification is easy.) It follows that an evaluation of the form $\sum \mathbf{Card}(\boldsymbol{sat}(G, \varphi))$ is not an MS-evaluation, but is nevertheless H_A-inductively computable, with the help of

the auxiliary evaluations $\mathbf{Card}(\boldsymbol{sat}(G, \varphi))$. The corresponding algorithm is again linear with uniform cost measure.

An application of this result is the computation of the average cardinality of a set X satisfying a formula φ in G (assuming there is at least one) defined by

$$\mathbf{AverageCard}(\boldsymbol{sat}(G, \varphi)) = \frac{\sum \mathbf{Card}(\boldsymbol{sat}(G, \varphi))}{\mathbf{Card}(\boldsymbol{sat}(G, \varphi))}.$$

Note that the cost of computing $\mathbf{AverageCard}(\boldsymbol{sat}(G, \varphi))$ is the same as that of computing $\sum \mathbf{Card}(\boldsymbol{sat}(G, \varphi))$ since, for the latter, one need also to determine $\mathbf{Card}(\boldsymbol{sat}(G, \varphi))$. Note, also, that **AverageCard** is H_A-inductively computable without (apparently) being an MS-evaluation.

As examples of applications, one can compute the average cardinality of a maximal independent set of G or a maximal clique of G.

4.5. Set of cardinalities

Here, we consider $\mathbf{Setcard}(A) = \{\mathbf{Card}(\alpha) \mid \alpha \in A\}$. Hence, **Setcard** maps $\mathscr{P}_f(\mathscr{P}_f(D)) \to \mathscr{P}_f(\mathbb{N})$. It is a $(\cup, \circledcirc)$-homomorphism $\langle \mathscr{P}_f(\mathscr{P}_f(D)), \cup, \cup, \emptyset, \emptyset \rangle \to \langle \mathscr{P}_f(\mathbb{N}), \cup, +, \emptyset, \{0\} \rangle$, where for $N, M \subseteq \mathbb{N}$, $N + M = \{n + m \mid n \in N, m \in M\}$.

In a computation relative to a graph G, each attribute (see Section 3.1) is a set of integers $\subseteq \{0, 1, \ldots, \mathbf{Card}(D_G)\}$. It can be represented by a Boolean vector of length at most $\mathbf{Card}(D_G)$. It follows that η is bounded by $O(\mathbf{Card}(D_G)^2)$. The corresponding algorithms are of time complexity $O(|t|^3)$, where t is the input tree.

4.6. The universal evaluation

For completeness sake, we consider now the evaluation structure $\mathscr{D} = \langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \cup, \cup, \emptyset, \emptyset \rangle$ and the computation of the whole set $\boldsymbol{sat}(G, \varphi)$. We take $n = 1$ in order to simplify the presentation. The attributes range over $\mathscr{P}_f(\mathscr{P}_f(D_G))$, and can be implemented by 2-dimensional Boolean arrays of size $m \times 2^m$, where $m = \mathbf{Card}(D_G)$. This gives for η an upper bound of order $O(2^m)$. The corresponding algorithm is of time complexity $O(2^{|t|})$.

In certain cases, the computation of $\boldsymbol{sat}(G, \varphi)$ is tractable and useful, as we now explain by an example. For each graph G, let $f_G : D_G \to D_G$ be a partial function such that there is an MS-formula $\varphi(X, Y)$ satisfying:

$$(G, X, Y) \models \varphi \quad \text{iff} \quad X = \{x\} \text{ and } Y = \{f_G(x)\} \text{ for some } x \in D_G.$$

The computation of $\boldsymbol{sat}(G, \varphi)$ yields a table defining f_G.

Let us say that a set of pairs $A \subseteq \mathscr{P}_f(D_G)^2$ is *functional* if the following conditions hold:

(i) if $(\alpha, \beta) \in A$, then $\mathbf{Card}(\alpha), \mathbf{Card}(\beta) \leqslant 1$,

(ii) if (α, β) and $(\alpha, \beta') \in A$, then $\beta = \beta'$.

It is clear that $\mu(\textbf{\textit{sat}}(G, \varphi))$ (where μ is defined in Section 3.2) is a set of functional sets of pairs; hence, the data structure will have to store functional sets of pairs only if we use the improved algorithm of Section 3.2 (see the remark made there).

Clearly, a functional set of pairs A can be represented by a vector with index set $D_G \cup \{\emptyset\}$ and values in $D_G \cup \{\emptyset, \perp\}$ (where $\perp$ stands for undefined; the value of x is $\perp$ if $(\{x\}, \beta)$ belongs to A for no β).

The operations $A \uplus B$ and $A \circledcirc B$ can be performed in time $O(\textbf{Card}(D_G))$. It follows that $\textbf{\textit{sat}}(G, \varphi)$, i.e. the table for f_G, can be computed in time $O(|t|^2)$, where t defines G.

In many cases of practical interest, one need not compute the whole set $\textbf{\textit{sat}}(G, \varphi)$ but only one tuple $(\alpha_1, \ldots, \alpha_n)$ of it. One may require that this tuple is optimal in a certain sense, say be such that $\textbf{Card}(\alpha_i)$ is maximum or minimum (for some fixed i), over all tuples in $\textbf{\textit{sat}}(G, \varphi)$.

We now explain how a *choice function*, associating such a tuple with a given graph G, can be efficiently computed. Let G be given by a term t. At every node u of t labeled by f, we have

$$w_\varphi(u) = \theta_{f,\varphi}(w_{\psi_{1,1}}(u_1), \ldots, w_{\psi_{n,m_n}}(u_n)), \tag{1}$$

where $w_\varphi(u)$ is the attribute occurrence evaluating to $\textbf{\textit{sat}}(\textbf{val}(t/u), \varphi)$ and $\theta_{f,\varphi}$ is the corresponding decomposition operator, namely a $(\uplus, \circledcirc)$-polynomial by Corollary 2.8.

If for each u and φ, we assume $w'_\varphi(u)$ to satisfy

$$w'_\varphi(u) \subseteq \theta_{f,\varphi}(w'_{\psi_{1,1}}(u_1), \ldots, w'_{\psi_{n,m_n}}(u_n)), \tag{2}$$

as opposed to (1). One then gets

$$w'_\varphi(u) \subseteq \textbf{\textit{sat}}(\textbf{val}(t/u), \varphi) \tag{3}$$

by bottom-up induction on u in t and by using the monotonocity of $(\uplus, \circledcirc)$-polynomials for set inclusion.

Each time we use (2), we can choose $w'_\varphi(u)$ to be a singleton; then one obtains at the end $w'_\varphi(\varepsilon)$, a singleton reduced to a tuple in $\textbf{\textit{sat}}(G, \varphi)$, as desired.

If in each case we choose

$$(\alpha_1, \ldots, \alpha_n) \in \theta_{f,\varphi}(w'_{\psi_{1,1}}(u_1), \ldots, w'_{\psi_{n,m_n}}(u_n)) \tag{4}$$

such that $\textbf{Card}(\alpha_i)$ is maximal (or minimal) to form $w'_\varphi(u) = \{(\alpha_1, \ldots, \alpha_n)\}$, then we obtain at the end in $w'_\varphi(u)$ a tuple in $\textbf{\textit{sat}}(G, \varphi)$ with an ith component of maximal (minimal) cardinality.

This shows that *optimal sets* in the sense of [6], satisfying an MS-property, can be constructed in time $O(|t|^2)$ (and even in time $O(|t|)$ by the variant presented in [6], Theorem 1).

This construction of optimal sets extends easily to the case of weighted graphs, that we now discuss.

4.7. Weighted graphs

Many graph problems involve *weighted* graphs. Routing and network design are examples of such problems (see [18]). Formally, a weighted graph G is a graph equipped with a function $w: D_G \to \mathbb{R}$ associating a *weight* $w(x)$ with each edge or vertex x. The mapping w is called a *weight* function. To take a few examples, a weight may be a capacity, a distance, a cost or a probability.

We shall assume that w is a mapping from D to $\mathbb{R}$, where D is the countable set of which all sets V_G and E_G are subsets. If $G = G_1 \oplus G_2$ or $G = \sigma_\alpha(G_1)$, then G_1 and G_2 inherit the weights of G in a natural way. In the case of $G = \theta_{i,j}(G_1)$, we shall use the convention made just before Lemma 2.5: $V_G \subseteq V_{G_1}$, $src_G(i) = src_{G_1}(i)$ and $src_{G_1}(j) \notin V_G$ whenever $src_{G_1}(i) \neq src_{G_1}(j)$. It follows that all elements of D_{G_1} have their weight as in D_G, except possibly $src_G(j)$, the weight of which may be arbitrarily chosen or taken conventionally to be 0.

For every finite set $X \subseteq D$, we let

$$w(X) = \sum \{w(x) \mid x \in X\}.$$

Other evaluations can be defined in terms of weights, notably

$$\mathbf{MaxWeight}(A) = \mathbf{Max}\{w(X) \mid X \in A\},$$

$$\mathbf{MinWeight}(A) = \mathbf{Min}\{w(X) \mid X \in A\},$$

where $A \in \mathscr{P}_f(\mathscr{P}_f(D))$.

These mappings are nothing but generalizations of the ones used in Case 4.3. The relevant evaluation structures are $\langle \mathbb{R} \cup \{-\infty\}, \mathbf{Max}, +, -\infty, 0\rangle$ and $\langle \mathbb{R} \cup \{+\infty\}, \mathbf{Min}, +, +\infty, 0\rangle$, and the corresponding algorithms are linear (for uniform cost measure). They make it possible to express the following evaluations:

1. Length of a minimal Hamiltonian circuit in a graph (where each edge has a positive length; the length of a path is the sum of the lengths of its edges; the result is $+\infty$ if the graph is not Hamiltonian). This makes it possible to express the traveling salesman problem ([ND22]).

2. Length of a longest or a shortest simple path linking the first source to the second.

Another useful evaluation structure is $\langle \mathbb{R}_+ \cup \{+\infty\}, \mathbf{Max}, \mathbf{Min}, +\infty, 0\rangle$, by which one can formalize the evaluation of the maximal capacity of a path from the first to the second source, namely

$$\mathbf{Cap}(G) = \mathbf{Max}\{\mathbf{Min}\{w(e) \mid e \in X\} \mid X \in \mathbf{sat}(G, \varphi)\},$$

where $\varphi(X)$ is the formula already used in Example 2.11. In fact, if we let, for $A \subseteq \mathscr{P}_f(D)$,

$$\mathbf{MM}(A) = \mathbf{Max}\{\mathbf{Min}\{w(x) \mid x \in \alpha\} \mid \alpha \in A\},$$

we have

$$\mathbf{MM}(A \cup B) = \mathbf{Max}(\mathbf{MM}(A), \mathbf{MM}(B))$$

and

$$\mathbf{MM}(A \cup B) = \mathbf{Min}(\mathbf{MM}(A), \mathbf{MM}(B)).$$

The following $\mathcal{NP}$-complete problems can also be expressed:

– Steiner tree in graphs [ND12]. (Is there a subtree T of G including all vertices of a given set $R \subseteq V_G$ such that the sum of the weights of all edges of T is at most a given value m?)

4.8. *Evaluation formulas*

We now assume that, instead of one weight function $w: D \to \mathbb{R}$, we have n of them, $w_i: D \to \mathbb{R}$, not necessarily all different. We let $w_i(X) = \sum\{w_i(x) \mid x \in X\}$ for X finite, $X \subseteq D$.

Let $\varphi(X_1, \ldots, X_n)$ be a formula. Let G be a graph. For every $(X_1, \ldots, X_n) \in \mathcal{P}_f(D_G)^n$, we let

$$\theta(X_1, \ldots, X_n) = c_1 w_1(X_1) + \cdots + c_n w_n(X_n),$$

where $c_1, \ldots, c_n$ are real numbers. Hence, θ is a linear evaluation term in the sense of [3]. For every set of tuples A, we let

$$\Theta(A) = \{\theta(X_1, \ldots, X_n) \mid (X_1, \ldots, X_n) \in A\}$$

and

$$\Theta(G, \varphi) = \{\theta(X_1, \ldots, X_n) \mid (X_1, \ldots, X_n) \in \mathbf{sat}(G, \varphi)\}$$

$$= \Theta(\mathbf{sat}(G, \varphi)).$$

Since

$$\Theta(A \cup B) = \Theta(A) \cup \Theta(B),$$

$$\Theta(A \uplus B) = \Theta(A) + \Theta(B) := \{m + m' \mid m \in \Theta(A), m' \in \Theta(B)\},$$

$$\Theta(\emptyset) = \emptyset, \qquad \Theta(\{\emptyset\}) = \{0\},$$

we get that Θ: $\langle \mathcal{P}_f(\mathcal{P}_f(D)^n), \cup, \uplus, \emptyset, \{\emptyset\} \rangle \to \langle \mathcal{P}_f(\mathbb{R}), \cup, +, \emptyset, \{0\} \rangle$ is a $(\cup, \uplus)$-homomorphism.

Hence, one can compute $\Theta(G, \varphi)$ from t such that $G = \mathbf{val}(t)$ in time $O(2^{n|t|})$. (One does not have here polynomial time because one must, in general, record the values of $\Theta(A)$ for an exponential number of tuples of A.)

Let us now consider the special case where $c_1, \ldots, c_m \in \mathbb{R}_+$ and $w_i: D \to \mathbb{R}$. We let

$$\mathbf{Max}\,\Theta(A) = \mathbf{Max}\{\theta(\alpha) \mid \alpha \in A\},$$

$$\mathbf{Min}\,\Theta(A) = \mathbf{Min}\{\theta(\alpha) \mid \alpha \in A\}.$$

Then, we have

$$\mathbf{Max}\,\Theta(A\cup B)=\mathbf{Max}(\mathbf{Max}\,\Theta(A),\mathbf{Max}\,\Theta(B)),$$

$$\mathbf{Min}\,\Theta(A\cup B)=\mathbf{Min}(\mathbf{Min}\,\Theta(A),\mathbf{Min}\,\Theta(B))$$

and

$$\mathbf{Max}\,\Theta(A\uplus B)=\mathbf{Max}\,\Theta(A)+\mathbf{Max}\,\Theta(B),$$

$$\mathbf{Min}\,\Theta(A\uplus B)=\mathbf{Min}\,\Theta(A)+\mathbf{Min}\,\Theta(B),$$

from which it follows that $\mathbf{Max}\,\Theta(G,\varphi)$ and $\mathbf{Min}\,\Theta(G,\varphi)$ are computable in time $O(|t|)$ if $t\in M(F)$ defines G (where F is fixed and finite). The computation of $\mathbf{Max}\,\Theta(G,\varphi)$ or $\mathbf{Min}\,\Theta(G,\varphi)$ is called a *linear EMS extremum* problem in [3]. Hence, we obtain another proof of the result of [3], stating that a linear EMS extremum problem can be solved in linear time (with uniform cost measure) for graphs of tree-width at most some fixed k (given by their tree-decompositions of width at most k).

4.9. Ordered graphs

Let us recall that we assume that D is linearly ordered by $\leqslant_D$. (If D is a set of memory locations, it is linearly ordered in a natural way.) Given an MS-formula $\varphi(X)$ and a graph G (with $D_G\subseteq D$), one may consider the problem of computing the *lexicographically first maximal* set X such that $\varphi(X)$ holds in G, i.e. the unique set $X\subseteq D_G$ such that

(1) $G\models\varphi(X)$,

(2) if $Y\supset X$ (and $Y\neq X$), then $G\models\neg\varphi(Y)$ (maximality of X) and

(3) if X' also satisfies (1) and (2), then $X\leqslant_{\text{lex}}X'$ (since X and X' are subsets of a totally ordered set, one can order them in increasing order and compare them by the lexicographic order associated with $\leqslant_D$).

Such a set X will be denoted by $\mathbf{LFM}(G,\varphi)$.

The complexity of finding lexicographically first maximal sets satisfying certain properties has been investigated by Miyano [24]. We consider the mapping $\mathbf{lfm}:\mathscr{P}_f(\mathscr{P}_f(D))\to\mathscr{P}_f(D)\cup\{\perp\}$ such that $\mathbf{lfm}(A)$ is the lexicographically first maximal element of A, namely,

(1) $\mathbf{lfm}(A)\subset\alpha$ for no α in A and

(2) $\mathbf{lfm}(A)\leqslant_{\text{lex}}\alpha$ for every $\subseteq$-maximal element α of A.

If $A=\emptyset$, then $\mathbf{lfm}(A)=\perp$ (undefined).

We have the following two facts, holding for all nonempty sets A and B in $\mathscr{P}_f(\mathscr{P}_f(D))$ such that $A\uplus B$ and $A\uplus B$ are defined:

Fact 4.1. $\mathbf{lfm}(A\uplus B)=\mathbf{lfmax}(\mathbf{lfm}(A),\mathbf{lfm}(B))$.

Fact 4.2. $\mathbf{lfm}(A\uplus B)=\mathbf{lfm}(A)\cup\mathbf{lfm}(B)$.

The operation **lfmax** is defined as follows:

$$\begin{aligned}\mathbf{lfmax}(x, y) &= x \quad \text{if } y \subseteq x \text{ or } y = \bot \\ &= y \quad \text{if } x \subseteq y \text{ or } x = \bot \\ &= \text{the first element of } \{x, y\} \text{ with respect to the order } \leqslant_{\text{lex}} \text{ if} \\ &\quad x, y \text{ are both } \neq \bot \text{ and are incomparable for } \subseteq.\end{aligned}$$

Letting $X \cup \bot = \bot \cup X = \bot$, Facts 4.1 and 4.2 also hold when A or B (or both) are empty. Note that $\mathbf{lfm}(\{\emptyset\}) = \emptyset$. Hence, **lfm** is a $(\uplus, \circledcirc)$-homomorphism $\langle \mathscr{P}_f(\mathscr{P}_f(D)^n), \uplus, \circledcirc, \emptyset, \emptyset \rangle \to \langle \mathscr{P}_f(D) \cup \{\bot\}, \mathbf{lfmax}, \cup, \bot, \emptyset \rangle$.

It follows that, for every monadic second-order formula $\varphi(X)$, the evaluation **LFM** is inductively computable. Each attribute has a value $\subseteq D_G$ (or $\bot$) which can be represented by a Boolean vector of length $\mathbf{Card}(D_G)$. The basic operations can be performed in time $O(\mathbf{Card}(D_G))$. Hence, one obtains a global time complexity $O(|t^2|)$.

4.10. A deterministic choice function

By using a linear ordering $\leqslant_D$ on D, we now construct a choice function $\mathbf{Ch}_n : \mathscr{P}_f(\mathscr{P}_f(D)^n) \to \mathscr{P}_f(D)^n \cup \{\bot\}$, that is a $(\cup, \circledcirc)$-homomorphism from $\mathscr{D}$ into an appropriate evaluation structure.

From $\leqslant_D$ we deduce an ordering of $\mathscr{P}_f(D)$, denoted by $\ll_D$, such that

$$\alpha \ll_D \alpha' \text{ iff } \mathbf{Card}(\alpha) < \mathbf{Card}(\alpha')$$

or

$$\mathbf{Card}(\alpha) = \mathbf{Card}(\alpha') \text{ and } \alpha \leqslant_{\text{lex}} \alpha',$$

where $\leqslant_{\text{lex}}$ is as in Section 4.9. We let then, for $A \subseteq \mathscr{P}_f(D)$,

$$\begin{aligned}\mathbf{prem}(A) &= \text{the unique } \ll_D\text{-minimal element of } A \\ &= \bot \text{ (undefined) if } A = \emptyset.\end{aligned}$$

We have

$$\mathbf{prem}(A \circledcirc B) = \mathbf{prem}(A) \cup \mathbf{prem}(B)$$

(with $\alpha \cup \bot = \bot \cup \alpha = \bot$ for all $\alpha \in \mathscr{P}_f(D) \cup \{\bot\}$) and

$$\mathbf{prem}(A \cup B) = \mathbf{p}(\mathbf{prem}(A), \mathbf{prem}(B)),$$

where

$$\mathbf{p}(\bot, \alpha) = \mathbf{p}(\alpha, \bot) = \alpha,$$

$$\mathbf{p}(\alpha, \beta) = \mathbf{prem}(\{\alpha, \beta\}) \quad \text{if } \alpha, \beta \neq \bot.$$

This function is the desired choice function for $n = 1$. For $n > 1$, we let

$$\begin{aligned}\mathbf{Ch}_n(A) &= \bot \quad \text{if } A = \emptyset \\ &= (\alpha_1, \alpha_2, \ldots, \alpha_n),\end{aligned}$$

where

$$\alpha_1 = \mathbf{prem}(\{\alpha \mid (\alpha, \beta_2, \ldots, \beta_n) \in A \text{ for some } \beta_2, \ldots, \beta_n\})$$

and

$$(\alpha_2, \ldots, \alpha_n) = \mathbf{Ch}_{n-1}(\{(\beta_2, \ldots, \beta_n) \mid (\alpha_1, \beta_2, \ldots, \beta_n) \in A\}).$$

It is easy to establish that

$$\mathbf{Ch}_n(A \cup B) = \mathbf{c}_n(\mathbf{Ch}_n(A), \mathbf{Ch}_n(B)),$$

where $\mathbf{c}_n(\bot, \alpha) = \mathbf{c}_n(\alpha, \bot) = \alpha$ and $\mathbf{c}_n(\alpha, \beta) = \mathbf{Ch}_n(\{\alpha, \beta\})$ if $\alpha, \beta \neq \bot$.

We also have

$$\mathbf{Ch}_n(A \uplus B) = \mathbf{Ch}_n(A) \cup \mathbf{Ch}_n(B)$$

$$= (\alpha_1 \cup \beta_1, \ldots, \alpha_n \cup \beta_n),$$

where $\mathbf{Ch}_n(A) = (\alpha_1, \ldots, \alpha_n)$ and $\mathbf{Ch}_n(B) = (\beta_1, \ldots, \beta_n)$.

It follows that $\mathbf{Ch}_n(\boldsymbol{sat}(G, \varphi))$ can be computed in time $O(|t|^2)$, where t defines G.

4.11. The evaluation of definable graph transductions

A *graph transduction* is a multivalued mapping from L to L', where L and L' are two sets of graphs. Courcelle has shown in [15, 16] that MS-formulas can be used to specify such transductions, called *monadic second-order definable graph transductions.*

There are numerous examples of interesting transductions: the mapping **val** from a tree t in $M(H_A)$ to the graphs denoted by t, and the "parsing" mapping, that associates with a graph G its derivation trees relative to a fixed "regular" hyperedge replacement grammar (as defined in [15]). See [16] for other examples and a survey of MS-definable transductions.

We now recall the definition. It is simpler to formulate it in terms of logical structures. Let R be a finite ranked set of relation symbols. The rank of a symbol r in R is denoted by $\rho(r)$. An R-*(relational) structure* is a tuple $S = \langle D_S, (r_S)_{r \in R} \rangle$, where D_S is a finite set, called the *domain* of S, and r_S is a subset of $D_S^{\rho(r)}$ for each r in R. We denote by $\mathscr{S}(R)$ the class of R-structures. Let W be a set of set variables, called here the set of parameters. (It is not a loss of generality to assume that all parameters are set variables.) We denote by $\mathscr{L}(R, W)$ the set of all MS-formulas that have their free variables in W.

Let R and R' be two ranked sets of relation symbols. An (R', R)-*definition scheme* is a tuple of formulas of the form

$$\Delta = (\varphi, \psi_1, \ldots, \psi_k, (\theta_w)_{w \in R' * k}),$$

where

$$k > 0,\ R' * k := \{(r, j) \mid r \in R',\ j \in [k]^{\rho(r)}\},$$

$$\varphi \in \mathscr{L}(R, W),$$

$$\psi_i \in \mathscr{L}(R, W \cup \{x_1\}) \qquad \text{for } i = 1, \ldots, k,$$

$$\theta_w \in \mathscr{L}(R, W \cup \{x_1, \ldots, x_{\rho(r)}\}) \quad \text{for } w = (r, j) \in R' * k.$$

Let $S \in \mathscr{S}(R)$ and let γ be a W-assignment in S. An R'-structure S' with domain $D_{S'} \subseteq D_S \times [k]$ is *defined by* Δ in (S, γ) if

$$(S, \gamma) \models \varphi,$$

$$D_{S'} = \{(d, i) \mid d \in D_S,\ i \in [k],\ (S, \gamma, d) \models \psi_i\},$$

for each r in R'

$$r_{S'} = \{((d_1, i_1), \ldots, (d_t, i_t)) \mid (S, \gamma, d_1, \ldots, d_t) \models \theta_{(r, j)}\},$$

where $j = (i_1, \ldots, i_t)$ and $t = \rho(r)$.

Note that S' is associated in a unique way with S, γ and Δ if it is defined, i.e. if $(S, \gamma) \models \varphi$. Hence, we can use the functional notation

$$S' = \mathbf{def}_\Delta(S, \gamma).$$

The transduction defined by Δ is the relation $\mathbf{def}_\Delta := \{(S, S') \mid S' = \mathbf{def}_\Delta(S, \gamma)$ for some W-assignment γ in $S\} \subseteq \mathscr{S}(R) \times \mathscr{S}(R')$. A transduction $f \subseteq \mathscr{S}(R) \times \mathscr{S}(R')$ is *definable* if it is equal to $\mathbf{def}_\Delta$ for some (R', R)-definition scheme Δ. In the case where $W = \emptyset$, we say that f is *definable without parameters* (it is functional).

A binary relation on graphs $f \subseteq \mathbf{G}_n(A) \times \mathbf{G}_m(A)$ is a *definable transduction* iff the relation on structures $\{(|G|, |G'|) \mid (G, G') \in f\}$ is definable (by some definition scheme Δ of appropriate type).

Proposition 4.1. *Let $f \subseteq \mathbf{G}_n(A) \times \mathbf{G}_m(A)$ be a definable transduction, and F be a finite subset of H_A. If t in $M(F)$ denotes $G \in \mathbf{G}_n(A)$, then one can do the following in polynomial time:*

(1) *decide whether $f(G) = \emptyset$;*

(2) *if $f(G) \neq \emptyset$, construct a graph G' in $f(G)$.*

Proof. (1) It follows from [14] that one can decide in polynomial time whether $G \models \exists Y_1, \ldots, Y_m \varphi$. (We let $Y_1, \ldots, Y_m$ be the parameters.)

(2) If $G \models \exists Y_1, \ldots, Y_m \varphi$, then one can construct in polynomial time a sequence of sets $(Y'_1, \ldots, Y'_m)$ satisfying φ. From these sets, the object structure $\mathbf{def}_\Delta(|G|_2, (Y'_1, \ldots, Y'_m))$ can be constructed in polynomial time by the results of the preceding sections. □

5. Hyperedge replacement graph grammars and compatible functions

In the present section, we compare our results with those of [20].

Let Γ be a hyperedge replacement grammar, and f be a function from graphs to values. This function is *Γ-compatible* [20] if there exists a finite sequence of functions

$f_0 = f, f_1, \ldots, f_n$ such that, for every i and for every G, if G is obtained by a derivation sequence

$$u \to_\Gamma H \overset{*}{\to}_\Gamma G,$$

where H has nonterminals $u_1, \ldots, u_m$, then

$$f_i = h(f_{j_1}(G_{j'_1}), \ldots, f_{j_m}(G_{j'_m})),$$

where $u_i \overset{*}{\to}_\Gamma G_i$ and $G = H[G_1/u_1, \ldots, G_m/u_m]$ (and h is a fixed mapping associated with H).

In our words, this means that f is inductively computable with respect to the graph operations associated with the right-hand sides of the grammar. (In the above definition, the graph operation associated with H maps $(G_1, \ldots, G_m)$ to $H[G_1/u_1, \ldots, G_m/u_m]$, the result of the simultaneous substitution of G_i for each u_i in H. See [5, 13] for more details.)

It follows that every inductively computable evaluation and, in particular, every MS-evaluation is Γ-compatible for every hyperedge replacement grammar Γ.

The following compatible functions are considered in [20]:

- number of nodes and number of hyperedges,
- number of simple paths from one source to another,
- maximal and minimal length of a simple path,
- number of simple cycles,
- maximal and minimal length of a simple cycle and
- maximal and minimal degree.

All these functions are MS-evaluations as we have seen above (the case of cycles is an easy extension of that of paths).

If f is an evaluation, from graphs to integers, then one can consider the following "boundedness" decision problems, where $n \in \mathbb{N}$ and Γ is a hyperedge replacement grammar, both given as inputs:

(1) Does there exist $G \in L(\Gamma)$ such that $f(G) \leqslant n$ (or $\geqslant n$ or $= n$)? More difficult seems to be:

(2) Does there exist m such that $f(G) \leqslant m$ for all $G \in L(\Gamma)$?

It is proved in [20] that problems of form 1 are decidable for evaluations such that all the decomposition operators can be written with $+$, $\times$, **max**, **min**, and that problems of form 2 are decidable too for those using only $+$, $\times$, **max**.

It follows that problems of forms 1 and 2 are decidable for evaluations of the forms **Card**(***sat***$(\cdot, \varphi)$), **Max Card**(***sat***$(\cdot, \varphi)$) and $\sum$**Card**(***sat***$(\cdot, \varphi)$), and that problems of form 1 are also solvable for those of the form **Min Card**(***sat***$(\cdot, \varphi)$), where, of course, φ is an MS-formula.

Acknowledgment

We thank H.J. Kreowski, S. Miyano and T. Nishizeki for helpful remarks and suggestions on the topics of this paper.

References

[1] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Algebraic Discrete Methods* **8** (1987) 277–287.

[2] S. Arnborg, B. Courcelle, A. Proskurowski and D. Seese, An algebraic theory of graph reduction, Report 90–92, Bordeaux-1 University, 1990. Short version in Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 70–83.

[3] S. Arnborg, J. Lagergren and D. Seese, Easy problems for tree decomposable graphs, *J. Algorithms* **12** (1991) 308–340.

[4] S. Arnborg and A. Proskurowski, Characterization and recognition of partial 3-trees, *SIAM J. Algebraic Discrete Methods* **7** (1986) 305–314.

[5] M. Bauderon and B. Courcelle, Graph rewriting and graph expressions, *Math. Systems Theory* **20** (1987) 83–127.

[6] J.A. Bern, E. Lawler and A. Wong, Linear time computation of optimal subgraphs of decomposable graphs, *J. Algorithms* **8** (1987) 216–235.

[7] H.L. Bodlaender, Dynamic programming algorithms on graphs with bounded tree-width, in: *Proc. 15th ICALP*, Lecture Notes in Computer Science, Vol. 317 (Springer, Berlin, 1988) 223–232.

[8] H.L. Bodlaender, Improved self-reduction algorithms for graphs with bounded tree-width, Tech. Report RUU-CS-88-29, University of Utrecht, 1988.

[9] H.L. Bodlaender, Complexity of path forming games, RUU-CS-89-29, Utrecht University, 1989.

[10] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *J. Algorithms* **11** (1990) 631–643.

[11] H.L. Bodlaender and T. Kloks, Better algorithms for path-width and tree-width of graphs, in: *ICALP '91*, Lecture Notes in Computer Science, Vol. 510 (Springer, Berlin, 1991).

[12] R.B. Borie, R.G. Parker and C.A. Tovey, Automatic generation of linear algorithms from predicate calculus descriptions of problems on recursively constructed graph families, *Algorithmica*, to appear.

[13] B. Courcelle, Graph rewriting: an algebraic and logic approach, in: J. van Leeuwen, ed., *Handbook of Computer Science, Vol. B* (Elsevier, Amsterdam, 1990) 193–242.

[14] B. Courcelle, The monadic second-order logic of graphs I: recognizable sets of finite graphs, *Inform. and Comput.* **85** (1990) 12–75.

[15] B. Courcelle, The monadic second-order logic of graphs V: on closing the gap between definability and recognizability, *Theoret. Comput. Sci.* **80** (1991) 53–202.

[16] B. Courcelle, Monadic second-order definable transductions, in: *CAAP '92*, Rennes, February 24–28, Lecture Notes in Computer Science, to appear, 1992.

[17] P. Deransart, M. Jourdan and B. Lorho, Attribute grammars, Lecture Notes in Computer Science, Vol. 323 (Springer, Berlin, 1988).

[18] M.R. Garey and D.S. Johnson, *Computers and Intractability* (W.H. Freeman and Company, San Francisco, 1979).

[19] A. Habel, Hyperedge replacement: grammars and languages, Ph.D. Thesis, University of Bremen, 1989.

[20] A. Habel, H.J. Kreowski and W. Vogler, Decidable boundness problems for hyperedge replacement graph grammars, Lecture Notes in Computer Science, Vol. 351 (Springer, Berlin, 1989) 275–289.

[21] W. Hohberg and R. Reischuk, A framework to algorithms for optimization problems on graphs, Technical Report, Technische Hochschule Darmstadt, Germany, 1990.

[22] D.S. Johnson, The NP-completeness column: an ongoing guide (16th), *J. Algorithms* **6** (1985) 434–451.

[23] J. Lagergren, Efficient parallel algorithms for tree-decomposition and related problems, in: *Proc. IEEE Symp. on FOCS* (1990) 173–182.

[24] S. Miyano, The lexicographically first maximal subgraph problems: P-completeness and NC algorithms, *Math. Systems Theory* **22** (1989) 47–73.

[25] K. Takamizawa, T. Nishizeki and N. Saito, Linear-time computability of combinatorial problems on series–parallel graphs, *J. Assoc. Comput. Mach.* **29** (1982) 623–641.

[26] J.W. Thatcher and J.B. Wright, Generalized finite automata theory with an application to a decision problem in second-order logic, *Math. Systems Theory* **2** (1968) 57–81.

[27] J. Valdes, E. Lawler and R. Tarjan, The recognition of series–parallel digraphs, *SIAM J. Comput.* **11** (1982) 289–313.

[28] T. Wimer, Linear algorithms on k-terminal graphs, Ph.D. Thesis, Clemson University, 1987.