Regular Expressions for Tree-Width 2 Graphs

Amina Doumane
CNRS, LIP, ENS Lyon, France

Abstract

We propose a syntax of regular expressions, which describes languages of tree-width 2 graphs. We show that these languages correspond exactly to those languages of tree-width 2 graphs, definable in the counting monadic second-order logic (CMSO).

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Tree width, MSO, Regular expressions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.121

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Acknowledgements I want to thank Denis Kuperberg for helpful discussions on the content and presentation.

1 Introduction

Regular word languages form a robust class of languages. One of the witnesses for this robustness is the variety of equivalent formalisms defining them. They can be described by finite automata, by monadic second-order (MSO) formulas, by regular expressions or by finite monoids [3, 6, 10]. Each of these formalisms has some advantages, depending on the context where it is used. For example, MSO is close to natural language, regular expressions define regular languages via their closure properties, automata have good algorithmic properties and can be used as actual algorithms to decide membership in a language, etc. Similarly, regular tree languages have equivalent formalisms, for various kinds of trees [11, 13, 9].

We will here further generalize the structures considered, by moving to graphs of bounded tree-width. Intuitively, they can be thought of as "graphs which resemble trees". In this framework, we already know that counting MSO (CMSO), an extension of MSO with counting predicates, and recognizability by algebra are equivalent [1, 2], yielding a notion that could be called "regular languages of graphs of tree-width k". Engelfriet [7] also proposes a regular expressions formalism matching this class, but by his own admission, these expressions closely mimic the behavior of CMSO. The main feature missing in Engelfriet's regular expressions is a mechanism for iteration, which is the central operator for word regular expressions: the Kleene star.

In this paper, we propose a syntax of regular expressions for languages of tree-width 2 graphs, that follow more closely the spirit of regular expressions on words, using Kleene-like iterations. This constitutes a first step towards the long term objective of obtaining such expressions for languages of graphs of tree-width k. We believe the case of tree-width 2 is already a significant step in itself. Graphs of tree-width 2 form a robust class of graphs with several interesting characterizations. One of them is the characterization via the forbidden minor K_4, the complete graph with four vertices. By the Robertson-Seymour theorem [12], it is known that for every $k \in \mathbb{N}$, the class of tree-width k graphs is characterized by a finite set of excluded minors. However, this result is not constructive, and only the forbidden minors for $k \leq 3$ are known.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 121; pp. 121:1–121:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let us now give more intuition about our expressions for graphs of tree-width 2. Our Kleene-like iteration is defined in terms of least fixed points $\mu x.\, e$. However without restriction, such an operator is too powerful and takes us outside of the CMSO-definable graphs. This phenomenon actually already happens on words: with an arbitrary fixed point, we can write $\mu x.\, (axb)$, defining the non-regular language $\{a^n x b^n \mid n \in \mathbb{N}\}$. The Kleene star on words can be seen as a restriction of the least fixed point operator: only fixed points of the form $\mu x.\, ex$ are allowed, where x does not appear in e. Here the idea is the same, but our restriction will be more involved, and will require a fine understanding of the structure of tree-width 2 graphs.

This work was inspired by the work of Gazdag and Németh [8] on regular expressions for bisemigroups and binoids. One of the main difference with our work is that their operators are only associative, while the operations generating our graphs satisfy more properties.

The paper is structured as follows. Sec. 2 is a preliminary section where we introduce graphs of tree-width 2, the logic CMSO and recognizability by algebra, which are known to be equivalent. In Sec 3, we introduce regular expressions, explain the condition that the iteration should satisfy, and give some examples to illustrate it. At the end of this section, we state our main result, which says that this formalism is equivalent to the two introduced in the preliminary section. We introduce in Sec. 4 the logic CMSO^r, an extension of CMSO with a very restricted form of quantification over relations, and show that it is equivalent to CMSO. Based on this, we show in section 5 that regularity implies CMSO-definability. Finally, we show in section 6 that recognizability implies regularity, proving our main result.

2 Preliminaries

Let Σ_1 and Σ_2 be two disjoint sets of *unary* and *binary* letters respectively. Throughout the paper, we work with the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$.

2.1 Tree-width 2 graphs

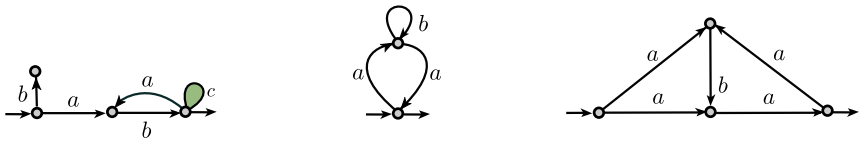
▶ **Definition 1** (Graphs). *A* graph G *is a tuple* $(V, E_1, E_2, s, t, l_1, l_2, \iota, o)$*, where* V *is a set of* vertices*,* E_1 *and* E_2 *are two disjoint sets of* unary *and* binary edges*,* $s : E_1 \uplus E_2 \to V$ *and* $t : E_1 \to V$ *are a* source *and a* target *functions specifying the source and the target of each edge*[1]*,* $l_1 : E_1 \to \Sigma_1$ *and* $l_2 : E_2 \to \Sigma_2$ *are* labeling functions *indicating the label of each edge,* ι *is the* input *vertex and* o *is the* output *vertex,* ι *and* o *are the* interface vertices *of* G*. All the vertices of* G *which are not interface vertices are called* inner vertices*. The* interface of G *is the pair* (ι, o) *if* $\iota \neq o$*, or the vertex* ι *otherwise. An* a-edge *is an edge labeled by the letter* a*. We say that* G *is* unary *if* $\iota = o$*, and* binary *otherwise. The* interface *of a binary edge* e *is* $(s(e), t(e))$*, the interface of a unary edge* e *is* $s(e)$*. An* interface in G *is a list of vertices of length* 1 *or* 2*. A graph is* empty *if it has no edges, and if all its vertices are interface vertices.*

▶ Remark 2. What we call here a graph is what is usually called a hypergraph (because of the unary edges) with interface. We depict graphs with unlabeled ingoing and outgoing arrows to denote the input and the output, respectively.

▶ **Definition 3** (Paths). *A* path p *of* G *is a non-repeating list* $(v_0, e_1, v_1, \ldots, e_n, v_n)$ *where* v_i *is a vertex of* G *and* e_i *is an edge of* G*, such that the interface of* e_i *is either* (v_{i-1}, v_i) *or* (v_i, v_{i-1})*, for every* $i \in [1, n]$*. The path* p *is* directed *if the interface of* e_i *is* (v_{i-1}, v_i) *for every* $i \in [1, n]$*. The vertex* v_0 *is the* input *of* p*,* v_n *is its* output *and* (v_0, v_n) *its* interface*. The path* p *is* safe *if it does not contain an interface vertex of* G *as an inner vertex.*

[1] For unary edges, we specify only their source.

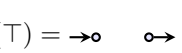
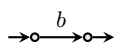
▶ **Example 4.** Here are some examples of graphs. The c-edge in the graph G a unary edge.



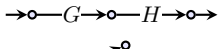
▶ **Definition 5** (tw_2 graphs). *Consider the signature σ containing the binary operations $\cdot$ and $\|$, the unary operations $^\circ$ and dom, and the nullary operations 1 and $\top$. We define* tw_2 terms *as the terms generated by the signature σ and the alphabet Σ:*

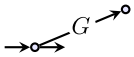
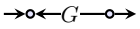
$$t, u := \ a \ \mid \ t{\cdot}u \ \mid \ (t \parallel u) \ \mid \ t^\circ \ \mid \ \mathsf{dom}(t) \ \mid \ 1 \ \mid \ \top \qquad\qquad a \in \Sigma$$

We define the graph of a term t, $\mathcal{G}(t)$, by induction on t, by letting:

$\mathcal{G}(1) =$ $\mathcal{G}(\top) =$  $\mathcal{G}(a) =$  $\mathcal{G}(b) =$ 

and interpreting the operations of the syntax as follows:

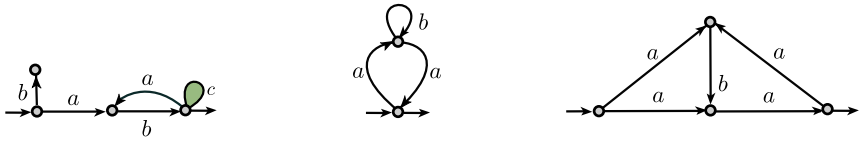
$G{\cdot}H =$  $G \parallel H =$ 

$\mathsf{dom}(G) =$  $G^\circ =$ 

In the picture above, we represent the graph G by an arrow from its input to its output. For example, the graph $\mathsf{dom}(G)$ is obtained from G by relocating the output to the input. We usually write tu instead of $t{\cdot}u$ and give priorities to the symbols of σ so that $ab \| c$ parses to $(a{\cdot}b) \| c$. We define the set of tw_2 graphs *as the graphs of the terms above. The graphs of a and $a \parallel 1$, where $a \in \Sigma$, are called* atomic.

We will sometimes identify terms with the graphs they generate. For example we may say that $(a \parallel b)$ is binary or connected to say that its graph is so.

▶ **Example 6.** Below, from left to right, two tw_2 graphs and a graph which is not tw_2.



$\mathsf{dom}(b)a(a^\circ \parallel b)c$ $\qquad$ $a(b \parallel 1)a$ $\qquad$ No corresponding term

▶ Remark 7. The tw_2 graphs are exactly those graphs whose skeleton[2] has tree-width 2 [4].

▶ **Definition 8** (Graph languages). *Sets of graphs are called* graph languages. *A graph language is* unary *or binary if all its graphs have this arity.*

2.2 Counting monadic second-order logic

We introduce CMSO, the *counting monadic second-order logic*, which is used to describe graph languages.

[2] The skeleton of a graph is the graph obtained by forgetting the labels and the orientation of the edges, and by adding an edge between the input and the output.

▶ **Definition 9** (The logic CMSO). *Let $\mathcal{V}$ be the relational signature which contains two binary symbols* source *and* target*, two unary symbols* input *and* output *and a unary symbol a for each (unary or binary) letter $a \in \Sigma$.*

Let $\mathbb{X}_1$ be a countable set of first-order variables *and $\mathbb{X}_2$ a countable set of* set variables *The formulas of* CMSO *are defined as follows:*

$$\varphi, \psi := \ r(x_1, \ldots, x_n) \ \mid \ x \in X \ \mid \ x = y \ \mid \ \exists x.\varphi \ \mid \ \exists X.\varphi \ \mid \ \varphi \vee \psi \ \mid \ \neg\varphi \ \mid \ (|X| \equiv k) \ [m]$$

where r is an n-ary symbol of $\mathcal{V}$, $x_1, \ldots, x_n, x \in \mathbb{X}_1$, $X \in \mathbb{X}_2$ and $k, m \in \mathbb{N}$. Free and bound variables *are defined as usual. A* sentence *is a formula without free variables. We use the usual syntactic sugar, for example $\varphi \Rightarrow \psi$ as a shortcut for $\neg\varphi \vee \psi$.*

We define the semantics of CMSO formulas. To handle free variables, CMSO formulas are interpreted over *pointed graphs*.

▶ **Definition 10** (Semantics of CMSO). *Let G be a graph and Γ be a set of variables. An* interpretation *of Γ in G is a function mapping each first-order variable of Γ to an edge or vertex of G, and each set variables to a a set of edges and vertices of G. A* pointed graph *is a pair $\langle G, I\rangle$ where G is a graph and I is an interpretation of a set of variables Γ in G. If Γ is empty, we denote it simply as G. Let φ be a* CMSO *formula whose free variables are Γ and let $\langle G, I\rangle$ be a pointed graph such that I is an interpretation of Γ. We define the* satisfiability relation $\langle G, I\rangle \models \varphi$ *as usual, by induction on the formula φ. Here is an example of the semantics of some* CMSO *formulas:*

$\mathsf{source}(v, e)$:	*the source of the edge e is the vertex v.*	$\mathsf{input}(v)$:	*v is the input of G.*
$\mathsf{target}(e, v)$:	*the target of the edge e is the vertex v.*	$\mathsf{output}(v)$:	*v is the output of G.*
$(\lvert X\rvert = k)[m]$:	*the size of X is congruent to k modulo m.*	$a(e)$:	*e is an a-edge.*

If φ is a sentence, we define $\mathcal{L}(\varphi)$, the graph language of φ *as follows:*

$$\mathcal{L}(\varphi) = \{G \ \mid \ G \text{ is a graph and } G \models \varphi\}.$$

▶ **Definition 11** (CMSO definability). *A graph language is* CMSO definable *if it is the graph language of a* CMSO *sentence.*

▶ **Example 12.** The language of graphs having an a-edge from the input to the output is definable in CMSO, by the following formula for instance:

$$\varphi := \ \exists e.\ \exists i.\ \exists o.\ \mathsf{input}(i) \wedge \mathsf{output}(o) \wedge a(e) \wedge \mathsf{source}(i, e) \wedge \mathsf{target}(e, o)$$

Note that the graphs of this language may not be tw_2 graphs.

▶ **Example 13.** The set of tw_2 graphs is a CMSO definable language. Indeed, tw_2 graphs are those graphs which exclude K_4, the complete graph with four vertices, as minor. The set of graphs which exclude a fixed set of minors can be easily defined in CMSO [5].

The set of tw_2 graphs having an a-edge from the input to the output is definable in CMSO, by the conjunction of the formula φ of Ex. 12 and the formula defining tw_2 graphs.

We state below a *localization result*, which allows us to transform a CMSO sentence into another one which talks only about a part of the original graph.

▶ **Proposition 14.** *Let φ be a* CMSO *sentence, $x, y \in \mathbb{X}_1$ and $X \in \mathbb{X}_2$. There is a* CMSO *formula $\varphi_{|(x,X,y)}$ such that, for every graph G and interpretation $I : (x \mapsto s, X \mapsto H, y \mapsto t)$, such that (s, H, t) is a subgraph of G, we have:*

$$\langle G, I\rangle \models \varphi|_{(x,X,y)} \qquad \Leftrightarrow \qquad (s, H, t) \models \varphi$$

▶ Remark 15. There is another presentation of the syntax of CMSO, where we remove first-order variables and the formulas including them, and add the following formulas:

$$X \subseteq Y \quad \text{and} \quad r(X_1 \dots, X_n) \quad \text{where } r \text{ is an } n\text{-ary symbol of } \mathcal{V}.$$

The formula $X \subseteq Y$ is interpreted as "X is a subset of Y" and $r(X_1 \dots, X_n)$ as "for each i, X_i is a singleton containing x_i and $r(x_1 \dots, x_n)$". This presentation is more convenient in proofs by induction as there are less cases to analyze.

2.3 Recognizability

We can specify languages of graphs by means of σ-algebras, generalizing to graphs the notion of recognizability by monoids. A *σ-algebra* $\mathcal{A}$ is the collection of a set D called its *domain*, and for each n-ary operation o of σ, a function $o^{\mathcal{A}} : D^n \to D$. A homomorphism $h : \mathcal{A} \to \mathcal{B}$ between two σ-algebras $\mathcal{A}$ and $\mathcal{B}$ is a function from the domain of $\mathcal{A}$ to the domain of $\mathcal{B}$ which preserves the operations of σ. Note that the set of tw_2 graphs, where the operations of σ are interpreted as in Def. 5, forms a σ-algebra which we denote by $\mathcal{G}_{\mathsf{tw}_2}$.

▶ **Definition 16** (Recognizability). *We say that a language L of tw_2 graphs is* recognizable *if there is a σ-algebra $\mathcal{A}$ with finite domain, a homomorphism $h : \mathcal{G}_{\mathsf{tw}_2} \to \mathcal{A}$ and a subset P of the domain of $\mathcal{A}$ such that $L = h^{-1}(P)$.*

▶ **Theorem 17.** *If a language of tw_2 graphs is CMSO definable, then it is recognizable.*

2.4 Operations on graph languages

The operations of σ can be lifted from graphs to graph languages in the natural way. We say that an operation on graph languages is CMSO *compatible* if, whenever its arguments are CMSO definable, then so is its result.

▶ **Proposition 18.** *Union and the operations of σ are CMSO compatible.*

We define two additional operations: *substitution* and *iteration*.

▶ **Definition 19** (Substitution and iteration). *Let x be a letter, L and M be tw_2 graph languages and let be G a tw_2 graph. We define the set of graphs $G[L/x]$ by induction on G as follows:*

$$x[L/x] = L, \quad a[L/x] = a \ (a \neq x) \quad \text{and} \quad o(G_1 \dots, G_n)[L/x] = o(G_1[L/x], \dots, G_n[L/x])$$

where o is an n-ary operation of σ. We define $M[L/x]$ as:

$$M[L/x] = \bigcup_{G \in M} G[L/x]$$

We define similarly the simultaneous substitution $M[\vec{L}/\vec{x}]$, where $\vec{L}$ and $\vec{x}$ are respectively a list of tw_2 graph languages and a list of letters of the same length.

For every $n \geq 1$, we define the language $L^{n,x}$ and the iteration $\mu x.L$ as follows:

$$L^{1,x} := L, \qquad L^{n+1,x} =: L[L^{n,x}/x] \cup L^{n,x}, \qquad \mu x.L := \bigcup_{n \geq 1} L^{n,x}.$$

▶ Remark 20. Substitution and iteration are not CMSO compatible in general. For instance, the iteration of the CMSO language $\{axb\}$, which is the set $\{a^n x b^n \mid n \in \mathbb{N}\}$, is not CMSO definable. However, under a *guard condition* that we introduce later, we recover CMSO compatibility.

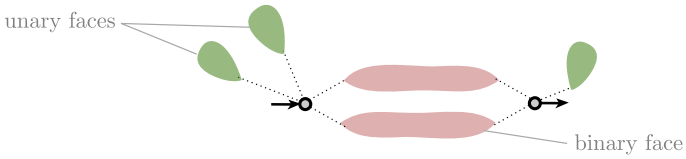
We finally consider two restricted forms of iteration called *Kleene* and *parallel iteration*.

▶ **Definition 21** (Kleene and parallel iteration). *We define the* Kleene iteration L^+ *and the* parallel iteration $L^{\|}$ *of a language L as follows, where x is a letter not appearing in L:*

$$L^+ = (\mu x.\ L{\cdot}x)[L/x], \qquad L^{\|} = (\mu x.\ L \,\|\, x)[L/x].$$

2.5 Pure graphs and modules

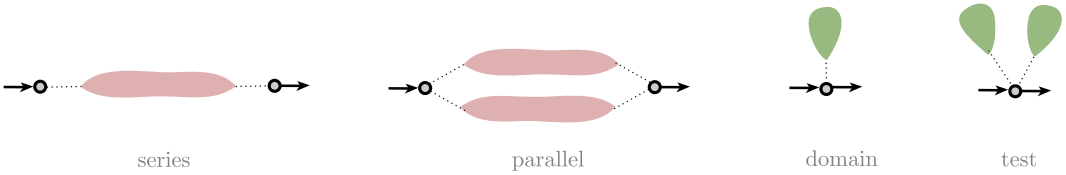
▶ **Definition 22** (Pure graphs.). *Let G be a graph. If we remove the interface vertices of G we obtain one or several connected components which we call the* faces *of G. The* arity of a face *is the number of interface vertices of G it is incident to.*



We say that G is pure *if it has at least one face and all it faces have the same arity as itself. We say that G is* prime *if it has exactly one face, and* composite *if it has at least two faces.*

▶ Remark 23. Pure graphs are connected and non-empty. Not all graphs are pure.

▶ **Definition 24** (Type of a pure graph). *The* type *of a pure graph is a pair specifying its arity and whether it is prime or composite. We say that a graph is* series *if it is binary and prime,* parallel *if it is binary and composite,* domain *if it is unary and prime and* test *if it is unary and composite. We denote by* $\mathsf{s}, \mathsf{p}, \mathsf{d}$ *and* t *the type series, parallel, domain and test respectively. Series, parallel domain and test graphs look like this:*



A graph language is (of type) series, parallel, domain *or* test *if* ***all*** *its graphs have this type.*

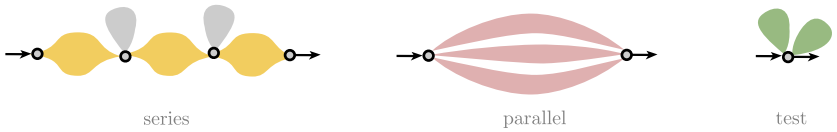
There is a canonical way to decompose pure graphs of type series, parallel and test.

▶ **Proposition 25** ([4]). *Let G be a pure graph. The graph G has the following shape:*

$$\begin{aligned} G &:= P_0 \cdot U_1 \cdot P_1 \ldots U_n \cdot P_n && \text{if } G \text{ is series,} \\ G &:= S_0 \,\|\, \cdots \,\|\, S_n && \text{if } G \text{ is parallel,} \\ G &:= D_0 \,\|\, \cdots \,\|\, D_n && \text{if } G \text{ is test,} \end{aligned}$$

P_j being parallel or atomic, U_i unary, S_i series and D_i domain, for all $j \in [0, n], i \in [1, n]$.

Here is a picture illustrating this proposition:

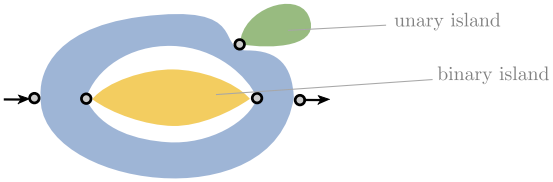


▶ **Definition 26** (Contexts). *Let* $\mathbb{S}$ *be a set of* special (unary and binary) letters*, and let* $n \geq 1$*. An* n-context *is a graph such that* n *of its edges, called* holes*, are numbered from* 1 *to* n*, and labeled by* n *distinct special letters. We call* 1*-contexts simply* contexts*.*

Let C *be an* n*-context whose holes are* $h_1 \dots, h_n$ *and let* $H_1, \dots, H_n$ *be graphs such that* h_i *and the* H_i *have the same arity, for all* $i \in [1, n]$*. We define* $C[H_1, \dots, H_n]$ *as the graph obtained from the disjoint union of* C *and* $H_1, \dots, H_n$*, by removing the holes of* G*, and for every* $i \in [1, n]$ *identifying the input of* h_i *with the input of* H_i*, the output of* h_i *with the output of* H_i*, and by letting its interface of to be that of* C*.*

▶ **Definition 27** (Islands and modules). *An* island *of a graph* G *is a graph* H *such that there is a context* C *satisfying* $G = C[H]$*. A* module *is a island which is pure. Two islands (or modules) of a graph are* parallel *if they have the same interface.*
Since modules are pure, we can speak of series, parallel, domain and test modules of a graph.

The following picture illustrates a unary and binary island of a graph.



▶ Remark 28. Our notion of modules is different from the one usually used in graph theory, more precisely in the setting of *modular decompositions*.

▶ Remark 29. The parallel composition of two islands of a graph G with the same interface is also an island of G with the same interface. Similarly, the parallel composition of two modules of a graph G with the same interface is also a module of G with the same interface. This justifies the following definition.

▶ **Definition 30** (Maximal islands and modules). *Let* G *be a graph and* I *an interface in* G*. The* maximal island at I *is the parallel composition of all the islands of* G *whose interface is* I*, we denote it by* $\mathsf{max\text{-}island}_G(I)$*. The* maximal module at I *is the parallel composition of all the modules of* G *whose interface is* I*, we denote it by* $\mathsf{max\text{-}module}_G(I)$*.*

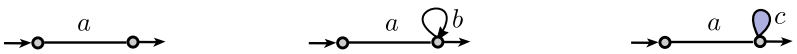
▶ Remark 31. The maximal module at a given interface does not always exist.

▶ **Proposition 32.** *Being series, parallel, domain, test, an island, a module, a maximal island, a maximal module are* CMSO *definable properties.*

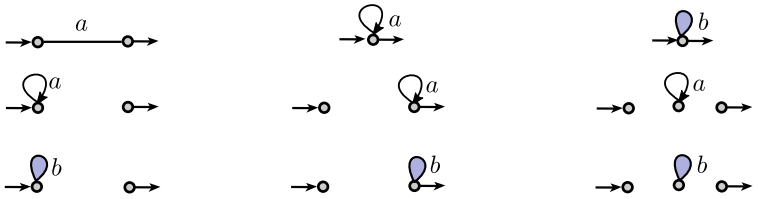
3 Regular expressions for tw_2 graphs

3.1 Regular expressions for word and multiset graphs

▶ **Definition 33** (Word and multiset alphabets). *Let* Σ_{w} *be the set of terms whose graphs have the following form, where* $a, b \in \Sigma_2$ *and* $c \in \Sigma_1$*:*



Let Σ_{m} *be the set of terms whose graphs have the following form, where* $a \in \Sigma_2$ *and* $b \in \Sigma_1$*:*



Word graphs *are the graphs generated from those of* Σ_{w} *by series composition, and* multiset graphs *are the graphs generated from those of* Σ_{m} *by parallel composition.*

▶ **Example 34.** Below, from left to right, a word graph and two multiset graphs.

▶ **Definition 35** (Word an multiset expressions)**.** Word expressions *are defined as follows:*

$$e, f := a \mid e \cdot f \mid e \cup f \mid e^{+} \qquad (a \in \Sigma_{\mathsf{w}})$$

Multiset pre-expressions *are defined as follows:*

$$e, f := a \mid (e \parallel f) \mid e \cup f \mid e^{\parallel} \qquad (a \in \Sigma_{\mathsf{m}})$$

Multiset expressions *are those pre-expressions, where each sub-term appearing under a parallel iteration, is built using a single element* $a \in \Sigma_{\mathsf{m}}$ *(all the other operations are allowed). The graph language of an expressions is defined as usual.*

▶ Remark 36. To see why the condition on multiset regular expressions is useful, consider the expression $e = (a \parallel b)$. The language of its parallel iteration is the set of multiset graphs which have the same number of a-edges and b-edges, and this is not a CMSO definable language.

3.2 Context-free expressions

▶ **Definition 37** (Context-free expressions)**.** *We define* context-free expressions *as the set of terms generated by the following syntax:*

$$\begin{aligned} e, f := \quad & e_{\mathsf{w}} \mid e_{\mathsf{m}} \\ & \mid e \cdot f \mid (e \parallel f) \mid e^{\circ} \mid \mathsf{dom}(e) \mid 1 \mid \top \\ & \mid e \cup f \mid e[f/x] \mid \mu x.e \end{aligned}$$

where e_{w} *and* e_{m} *are respectively word and multiset regular expressions. We define the* language *of a context-free expression* e*, denoted* $\mathcal{L}(e)$*, by induction on* e*, interpreting the operations of the syntax as described in Sec. 2.4.*

Regular expressions for tw_2 *graphs* will be defined as a restriction of context-free expressions, where substitution and iteration are allowed only under a *guard condition* that we shall explain in the following.

3.3 The guard condition

▶ **Definition 38** (Guarded letters)**.** *Let* G *be a graph and* x *a letter. We say that:*

- x *is* s-guarded *in* G *if* x *is binary and every* x*-labeled edge of* G *is parallel to a module.*
- x *is* p-guarded *in* G *if* x *is binary and no* x*-labeled edge of* G *is parallel to a module.*

- *x is* d*-guarded in G if x is unary.*
- *x is* t*-guarded in G if x is unary and no x-labeled edge of G is parallel to a module.*

Let $\tau \in \{\mathsf{s}, \mathsf{p}, \mathsf{d}, \mathsf{t}\}$ be a type and L a graph language. We say that x is τ-guarded *in L if it is τ-guarded in every graph of L.*

▶ **Definition 39** (Guard condition)**.** *Let x be a letter, M a tw_2 graph language and L a pure language of type τ. The substitution $M[L/x]$ is guarded if x is τ-guarded in M. The iteration $\mu x.L$ is guarded if x is τ-guarded in L.*

We say that the iteration $\mu x.L$ is of type τ *if L is of type τ.*

▶ **Definition 40** (Regular expression)**.** *A* regular expression *is a context-free expression where every substitution and iteration is guarded. A language of graphs is* regular *if it is the language of some regular expression.*

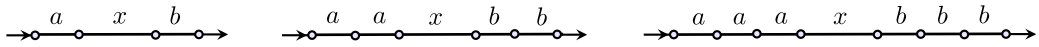
▶ Remark 41. When L is test and x is a unary letter, then $\mu x.L$ is always guarded.

▶ **Proposition 42.** *We can decide if a context-free expression is regular.*

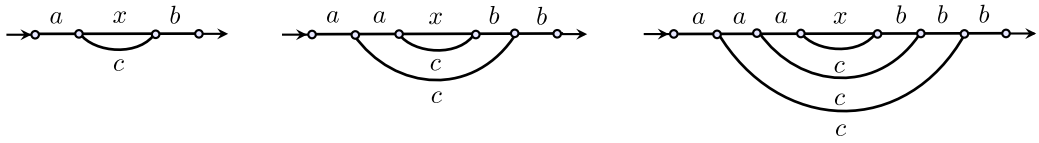
▶ Remark 43. Be aware that prop. 42 is about deciding a syntactic property of e, namely that the iterations and substitutions are guarded. However, the problem of determining if a context-free expression *defines a* CMSO *language* is undecidable. This apparent contradiction comes from the fact that some context-free expressions, which are not guarded, define CMSO languages, as we shall see in the upcoming examples.

3.4 Examples

▶ **Example 44.** The iteration $\mu x.axb$ is not guarded. Indeed, the language of axb is series, as it contains a single series graph G. However, the letter x is not s-guarded in G, because it is not parallel to any module of G. The graph of this iteration look like this:

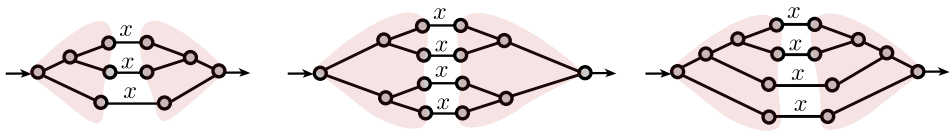


▶ **Example 45.** The iteration $\mu x.a(x \parallel c)b$ is guarded. Indeed the language of $a(x \parallel c)b$ is series, actually it contains a single graph G, depicted below left, which is series. The letter x is s-guarded in G, because it is parallel to a module, namely the c-edge. The graph of this iteration look like this:



Note the similarity between the graph language of $\mu x.axb$ and that of $\mu x.a(x \parallel c)b$: the former is obtained by forgetting the c-edges of the latter. Yet, the latter is CMSO definable, while the former is not. In the case of $\mu x.a(x \parallel c)b$, the c-edges will guide a CMSO formula to relate the a-edges and the b-edges of the same iteration depth. This is the main intuition behind the guard condition for series languages.

▶ **Example 46.** The iteration $\mu x.(axa \parallel axa)$ is guarded. Indeed, the language of $(axa \parallel axa)$ is parallel, as it contains a unique graph G (the left graph below) which is parallel. The letter x is p-guarded because all the occurrences of x are not parallel to any module of G. Note that the graphs of this expression have the following shape: they all start with a binary tree whose edges are labeled by a, end ends with the mirror image of this tree, while the corresponding leafs are connected by an x-edge. Those trees are colored in red below.



At first glance, this expression dos not seem to be CMSO definable, as it seems that we need to test whether a graph starts and ends with the same tree. We will see however that the language of this expression, as those of all regular expressions, is CMSO definable.

The guard condition is not "perfect", in the sense that some non-guarded context-free expressions might generate CMSO definable languages, as shown in the following example.

▶ **Example 47.** The context-free expression $(\mu x.axb)[1/a, 1/x, 1/b]$ is not regular because the iteration $\mu x.axb$ is not guarded. However its language, the graph of 1, is CMSO definable.

▶ Remark 48. Intuitively, the guard condition allows only those graphs where series and parallel operations alternate. This why we add the word and multiset expressions: to allow graphs where we can iterate only series or parallel operations respectively.

3.5 Main result

The main result of this paper is the following theorem:

▶ **Theorem 49.** *Let L be a language of* tw_2 *graphs. We have:*

$$L \text{ is recognizable} \quad \Leftrightarrow \quad L \text{ is CMSO definable} \quad \Leftrightarrow \quad L \text{ is regular}$$

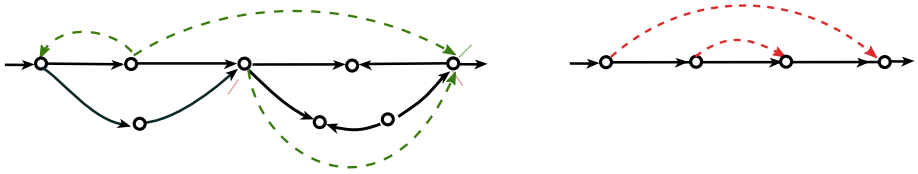
Thanks to Thm. 17, CMSO definability implies recognizablity. We show that regularity implies CMSO definability in Sec. 5 and that recognizabilty implies regularity in Sec. 6.

4 Companion relations

▶ **Definition 50** (Companion relation). *Let G be a graph. Two paths of G are* orthogonal *if they do not share any edge, and whenever they share a vertex, it is necessarily an interface vertex of one of them. A set of paths is a* set of orthogonal paths *if its paths are pairwise orthogonal.*

A relation R on the vertices of G is a companion relation *if there is a set of orthogonal paths P such that $(v, w) \in R$ iff (v, w) is the interface of a path $p \in P$. We say that p is a* witness *for (v, w), and that P is a* witness *for the relation R.*

▶ **Example 51.** The relation indicated by the green dotted arrows below is a companion relation. This is not the case for the one indicated by the red dotted arrows.



We introduce CMSO^r, an extension of CMSO where quantification over companion relations is possible.

▶ **Definition 52** (The logic CMSO^r). *Let $\mathbb{X}_r$ be a set of* relation variables*, whose elements are denoted $R, S, \ldots$. The formulas of CMSO^r are of the following form:*

$$\varphi := \mathsf{CMSO} \mid \exists R.\ \varphi \mid (x, y) \in R \qquad (R \in \mathbb{X}_r,\ x, y \in \mathbb{X}_1).$$

As for CMSO, we need to define the semantics of a formula over pointed graphs to handle free variables.

▶ **Definition 53** (Semantics of CMSO^r). *Let G be a graph and Γ be a set of variables. An interpretation of Γ is as usual, but here every relation variable is mapped to a binary relation on the vertices of G. We define the* satisfiability relation $\langle G, I\rangle \models \varphi$ *as usual, by induction on the formula φ. The only new cases are the quantification $\exists R$ which is interpreted as "there exists a companion relation R on the vertices of the graph", and the formulas $(x, y) \in R$ which are interpreted as "there is a pair of vertices (x, y) in R".*

4.1 The logic CMSO^r have the same expressive power as CMSO

To guess a companion relation in CMSO, we show how to encode a set of guarded paths by a collection of sets called a *footprint*.

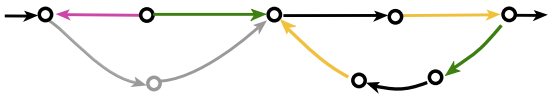
▶ **Definition 54** (Frontier edges of a path). *Let $p = (v_0, e_1, v_1, \ldots, e_n, v_n)$ be a path. If $n > 1$, we call e_1 the* opening edge *of p and e_n its* closing edge*. If $n = 0$, we call e_0 its* single edge*. Opening, closing and single edges are called the* frontier edges *of p, the other edges are called its* inner edges*.*

▶ **Definition 55** (Footprint). *A* footprint *in a graph G is the following collection of data: a partition of the vertices of G into* non-path *and* path *vertices, a partition of edges into* non-path *and* path *edges, a partition of path edges into* frontier *and* inner *edges, a partition of frontier edges into* opening, closing and single *edges and a partition of path edges into* direct *and* inverse *edges.*

The partition of path edges into direct and inverse ones provides them with a new orientation: they conserve their original orientation if they are direct, or get reversed (we swap the source and target) if they are inverse edges.

Let $\mathbb{F}$ be a footprint. A path p is encoded by $\mathbb{F}$ if its edges and vertices are path edges and path vertices of $\mathbb{F}$, if its inner, frontier, opening, closing and single edges are edges of the corresponding sets in $\mathbb{F}$. Moreover, p must form a directed path with the new orientation dictated by $\mathbb{F}$.

▶ **Example 56.** We represent below a footprint in the left graph of Ex. 51. Non-path edges and vertices are grey, path vertices are black, opening edges are green, closing edges are yellow, single edges are pink and all the other inner edges are black. For path edges, we display the new orientation induced by the footprint instead of the original one. The set of paths encoded by this footprint are a witness that the green relation of Ex. 51 is a companion relation.



▶ **Proposition 57.** *Let G be a graph and P a set of orthogonal paths of G. There is a footprint $\mathbb{F}$ such that P is the set of paths encoded by $\mathbb{F}$.*

▶ **Corollary 58.** *If a language is CMSO^r definable then it is CMSO definable.*

5 Regular implies CMSO definable

▶ **Theorem 59.** *If a language is regular, then it is* CMSO *definable.*

To prove Thm. 59, we proceed by induction on regular expressions. The cases of word and multiset regular expressions follow from the similar result for words and commutative words. The cases of union and the operations of the signature σ follow from Prop. 18. We are left with the cases of substitution and iteration; the rest of this section is dedicated to proving the following proposition.

▶ **Proposition 60.** *Let x be a letter and L and M be languages of* tw_2 *graphs. We have:*

$$M[L/x] \text{ is guarded and } L \text{ and } M \text{ are } \mathsf{CMSO}\text{-definable} \Rightarrow M[L/x] \text{ is } \mathsf{CMSO}\text{-definable.}$$
$$\mu x.L \text{ is guarded and } L \text{ is } \mathsf{CMSO}\text{-definable} \Rightarrow \mu x.L \text{ is } \mathsf{CMSO}\text{-definable.}$$

We handle the case of iteration, the case of substitution being similar. We show first that the iteration of a CMSO definable language, without any guard condition, is definable in an extension of CMSO where we are allowed to quantify existentially over sets of subgraphs of the input graph, which we call CMSO^d. This logic is obviously strictly more expressive then CMSO, because it amounts to quantify over sets of sets. Based on this, we show that the *guarded iteration* of a CMSO definable language is definable in CMSO^r, the extension of CMSO with companion relations defined in the previous section. This concludes the proof, the logic CMSO^r being equivalent to CMSO.

5.1 Iteration of CMSO formulas is CMSO^d definable

5.1.1 Decompositions

When a graph is in the iteration $\mu x.L$ of some language L, it is possible to structure it into a tree shaped decomposition, such that each part of this decomposition "comes from L". In the following, we define such decompositions.

▶ **Definition 61** (Independent graphs)**.** *Let G be a graph and H, K be subgraphs of G. We say that H and K are* independent *if they do not share any edge; and whenever they share a vertex, it is necessarily an interface vertex of both H and K.*

▶ **Definition 62** (Decompositions)**.** *A* decomposition *of G is a set $\mathcal{D}$ of modules of G such that $G \in \mathcal{D}$ and for every pair of graphs in $\mathcal{D}$, they are either independent, or module one of the other. We call the graphs of a decomposition its* components*. We call* the interfaces of $\mathcal{D}$ *the set of interfaces of its components.*

Let H and K be components of a decomposition $\mathcal{D}$. We say that H is a child *of K, if H is a module of K, and if there is no component C of $\mathcal{D}$, distinct from H and K, such that H is a module of C and C is a module of K.*

The graph G is called the head *of $\mathcal{D}$. A component of $\mathcal{D}$ is a* leave *if it does not contain another component of $\mathcal{D}$ as a module.*

▶ **Definition 63** (Body of a component)**.** *Let G be a graph, $\mathcal{D}$ a decomposition of G and C a component of $\mathcal{D}$.*

The body *of C is the subgraph of G whose vertices are those of C minus the* ***inner*** *vertices of its children; and whose edges are those of C minus those of its children.*

*The x-*body of C *is the graph whose interface is the interface of C, whose vertices are the vertices of the body of C, and whose edges are the edges of the body of C plus, for each child F of C, an x-edge whose interface is the interface of F. We denote it by x-$\mathsf{body}_{\mathcal{D}}(C)$.*

▶ **Definition 64** (L-decompositions). *Let L be a graph language. An L-decomposition of a graph G is a decomposition of G such that the x-body of each of its components is in L.*

▶ Remark 65. The body of a component is a subgraph of G, but its x-body is not a subgraph of G in general, because of the added x-edges.

▶ **Proposition 66.** *Let L be a graph language. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists \mathcal{D}. \quad \mathcal{D} \text{ is an } L\text{-decomposition of } G.$$

5.1.2 The logic CMSO^d

Let φ be a CMSO formula defining a graph language L. Using Prop 66, we can express that a graph G is in the iteration $\mu x.L$ by guessing a decomposition $\mathcal{D}$ of G, and ensuring that the x-body of each component satisfies φ. But guessing a set of subgraphs is not expressible in CMSO. This is why we introduce CMSO^d, an extension of CMSO where this is allowed.

▶ **Definition 67** (CMSO^d logic). *Let $\mathbb{X}_d$ be a set of* graph set variables*, whose elements are denoted $\mathcal{X}, \mathcal{Y} \ldots$. The formulas of* CMSO^d *are of the following form:*

$$\varphi := \mathsf{CMSO} \mid \exists \mathcal{X}.\ \varphi \mid (s, Z, t) \in \mathcal{X} \qquad (\mathcal{X} \in \mathbb{X}_d, \quad Z \in \mathbb{X}_2, \quad s, t \in \mathbb{X}_1).$$

Free and bound variables are defined as usual. As for CMSO, we need to define the semantics of a formula over pointed graphs to handle free variables.

▶ **Definition 68** (Semantics of CMSO^d). *Let G be a graph and Γ be a set of variables.*

An interpretation of Γ *is a function mapping every first-order variable of Γ to an edge or vertex of G, every set variable to a set of edges and vertices of G, and every graph set variable to a set of subgraphs of G.*

We define the satisfiability relation $\langle G, I \rangle \models \varphi$ *as usual, by induction on φ. The only new cases compared to* CMSO *are the quantification $\exists \mathcal{X}$ which is interpreted as "there exists a set of subgraphs $\mathcal{X}$", and the formulas $(s, Z, t) \in \mathcal{X}$ which are interpreted as "the graph whose input is s, whose output is t and whose set of edges and vertices is Z, is an element of $\mathcal{X}$".*

▶ **Proposition 69.** *There is a* CMSO^d *formula* $\mathsf{decomp}(\mathcal{X})$*,* without graph set quantification*, such that for every graph G and every set of subgraphs $\mathcal{D}$ of G, we have:*

$$\langle G, \mathcal{X} \mapsto \mathcal{D} \rangle \models \mathsf{decomp}(\mathcal{X}) \quad \Leftrightarrow \quad \mathcal{D} \text{ is a decomposition of } G.$$

5.1.3 Iteration is expressible in CMSO^d

Given a CMSO formula φ, we construct a formula $[\![\varphi]\!]$ having $\mathcal{X}$ as unique free variable, which expresses the fact that the x-body the head of the decomposition $\mathcal{X}$ satisfies φ. To construct $[\![\varphi]\!]$, we need the following definition.

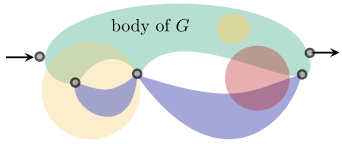
▶ **Definition 70** (Complete sets). *Let $\mathcal{D}$ be a decomposition of a graph G.*

Let H be a set of edges and vertices of G. We say that H is complete *if, whenever it contains an edge or an inner vertex of a child C of G (seen as a component of $\mathcal{D}$), then it contains all the edges and inner vertices of C.*

Let K be a set of edges and vertices of the x-body of G. We denote by $\mathsf{complete}_{\mathcal{D}}(K)$ the set of edges and vertices of G, obtained from K by replacing every x-edge coming from a child C of G by the set of edges and inner vertices C.

▶ Remark 71. Note that if H is complete, there is a set S such that $H = \mathsf{complete}_{\mathcal{D}}(S)$.

Here is a picture illustrating complete sets. The green part is the body of G and the purple modules are its children. The yellow sets are complete, but the pink one is not.



▶ **Proposition 72.** *The following formulas are* CMSO^d *definable:*

$\mathsf{child}_{\mathcal{X}}(Y)$	:	*Y is the set of edges and inner vertices of a child of the input graph w.r.t. the decomposition $\mathcal{X}$.*
$\mathsf{is\text{-}complete}_{\mathcal{X}}(Y)$	:	*Y is complete wrt $\mathcal{X}$.*
$\mathsf{body\text{-}edge}_{\mathcal{X}}(Y)$	:	*Y is a singleton containing an edge from the body of the input graph wrt $\mathcal{X}$.*
$\mathsf{source}_{\mathcal{X}}(Y,Z)$	:	$\mathsf{child}_{\mathcal{X}}(Z)$ *and Y is a singleton containing the input of the corresponding child.*
$\mathsf{target}_{\mathcal{X}}(Y,Z)$	:	*the same as above, where input should be replaced by output.*
$\mathsf{choice}_{\mathcal{X}}(Y,Z)$	:	*Z contains all the body elements of Y, and for every child contained in Y, Z contains exactly one element of this child.*

We construct the formula $[\![\varphi]\!]$ by induction on the structure of φ. We suppose that φ is build using the syntax of CMSO where only set variables are allowed.

▶ **Definition 73.** *Let φ be a* CMSO *formula whose free variables are Γ. We define the* CMSO^d *formula $[\![\varphi]\!]$, whose free variables are $\Gamma \cup \{\mathcal{X}\}$, by induction as follows:*

$$
\begin{array}{lcl}
[\![\varphi \vee \psi]\!] & = & [\![\varphi]\!] \vee [\![\psi]\!] \\
[\![\neg\varphi]\!] & = & \neg\, [\![\varphi]\!] \\
[\![(|Y| \equiv k)[m]]\!] & = & \exists Z.\ \mathsf{choice}_{\mathcal{X}}(Y,Z) \wedge (|Z| \equiv k)[m] \\
[\![Y \subseteq Z]\!] & = & Y \subseteq Z \\
[\![a(Y)]\!] & = & a(Y) \qquad (a \neq x) \\
[\![x(Y)]\!] & = & \mathsf{child}_{\mathcal{X}}(Y) \vee (\mathsf{body\text{-}edge}_{\mathcal{X}}(Y) \wedge x(Y)) \\
[\![\exists Y.\ \varphi]\!] & = & \exists Y.\ \mathsf{is\text{-}complete}_{\mathcal{X}}(Y) \wedge [\![\varphi]\!] \\
[\![\mathsf{source}(Y,Z)]\!] & = & (\mathsf{body\text{-}edge}_{\mathcal{X}}(Z) \wedge \mathsf{source}(Y,Z)) \vee (\mathsf{child}_{\mathcal{X}}(Z) \wedge \mathsf{source}_{\mathcal{X}}(Y,Z)) \\
[\![\mathsf{target}(Y,Z)]\!] & = & (\mathsf{body\text{-}edge}_{\mathcal{X}}(Z) \wedge \mathsf{target}(Y,Z)) \vee (\mathsf{child}_{\mathcal{X}}(Z) \wedge \mathsf{target}_{\mathcal{X}}(Y,Z))
\end{array}
$$

Transfer results are results of this form: to check that a transformation $\mathsf{f}(G)$ of a structure G satisfies a formula φ, construct a formula $\mathsf{f}^{-1}(\varphi)$ that G should satisfy. The proposition below is a transfer result, where the transformation is the x-body.

▶ **Proposition 74.** *Given a* CMSO *sentence φ defining, there is a* CMSO^d *formula $[\![\varphi]\!]$ having $\mathcal{X}$ as unique free variable, such that for every graph G and every decomposition $\mathcal{D}$ of G* whose components are non-empty*:*

$$\langle G, \mathcal{X} \mapsto \mathcal{D} \rangle \models [\![\varphi]\!] \qquad \Leftrightarrow \qquad x\text{-}\mathsf{body}_{\mathcal{D}}(G) \models \varphi.$$

The formula $[\![\varphi]\!]$ expresses the fact that the x-body of the head of a decomposition satisfies φ. Using this formula and the localization construction of Prop. 14, we construct a formula $\mu x.L$ saying that the x-body of **all** the components of a decomposition satisfy φ.

▶ **Definition 75.** *If φ is a* CMSO *formula, we let $\mu x.\varphi$ be the following* CMSO^d *formula:*

$$\mu x.\varphi := \exists \mathcal{X}.\ \mathsf{decomp}(\mathcal{X}) \quad \wedge \quad \forall s.\forall Z.\ \forall t.\ (s,Z,t) \in \mathcal{X} \ \Rightarrow\ [\![\varphi]\!]\,|_{(s,Z,t)}$$

The following proposition says that the language of $\mu x.\varphi$ is the iteration of that of φ.

▶ **Proposition 76.** *If φ is a* CMSO *formula defining a language of non-empty graphs, then:*

$$\mathcal{L}(\mu x.\varphi) = \mu x.\mathcal{L}(\varphi).$$

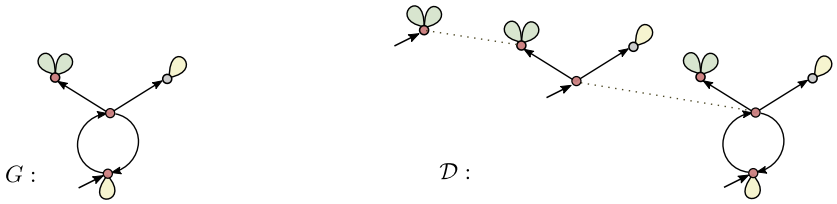
▶ **Corollary 77.** *If L is* CMSO *definable then $\mu x.L$ is* CMSO^d *definable.*

5.2 Guarded iteration of CMSO languages is CMSO^r

The idea here is that when the iteration $\mu x.L$ is guarded, L-decompositions can be encoded by sets of edges and vertices and by companion relations.

5.2.1 The case of test languages

Let $\mu x.L$ be a guarded iteration of type *test*, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. Suppose that G is the left graph below, and that the red vertices are the interfaces[3] of $\mathcal{D}$.



We claim that, thanks to the guard condition, this information is enough to reconstruct the whole decomposition $\mathcal{D}$. More precisely, we claim that the components of $\mathcal{D}$ are exactly the maximal modules of G, whose interfaces are the red vertices, as depicted above.

▶ **Definition 78.** *Let G be a graph and S be a set of vertices of G. We define $\mathcal{D}_{\mathsf{t}}(S)$ as the set of maximal modules of G, whose type is test, and whose interfaces belong to S.*

▶ **Proposition 79.** *Let $\mu x.L$ be a guarded iteration of type test. We have:*

$$G \in \mu x.L \qquad \Leftrightarrow \qquad \exists S.\ \ S \text{ is a set of vertices of } G \text{ and } \mathcal{D}_{\mathsf{t}}(S) \text{ is an } L\text{-decomposition of } G.$$

Proof. ($\Rightarrow$) Follows from Prop. 66. To prove ($\Leftarrow$), we define the property P_n as follows:

$$P_n : \qquad \forall G.\ \ G \in L^{n,x} \ \Rightarrow\ \exists S.\ \ S \text{ is a set of vertices of } G \text{ and } \mathcal{D}_{\mathsf{t}}(S) \text{ is an } L\text{-decomposition of } G.$$

We prove, by induction on n, that P_n is valid for every $n \geq 1$, and this is enough to conclude.

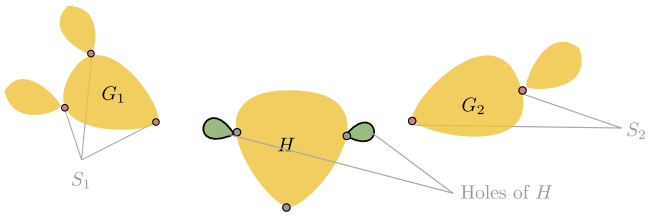
[3] Recall that test graphs are unary, hence all the components of a decomposition of G are unary.

When $n = 1$, take S to be the singleton containing the interface of G. We have that $\mathcal{D}_{\mathsf{t}}(S) = \{G\}$ and since $G \in L$, we have that $\mathcal{D}_{\mathsf{t}}(S)$ is an L-decomposition of G.

Let $G \in L^{n+1,x}$. By definition, there is a k-context H and graphs $G_1, \dots, G_k$ such that:

$$G = H[G_1, \dots, G_k], \quad H[x, \dots, x] \in L \quad \text{and} \quad G_i \in L^{n,x}, \text{ for } i \in [1, k].$$

Thanks to the guard condition, there is no module of H parallel to a hole of H. For every $i \in [1, k]$, let S_i be the set of vertices provided by the induction hypothesis applied to the graph G_i. Here is a picture illustrating these notations:



The set of subgraphs $\mathcal{D}$ defined below is an L-decomposition of G.

$$\mathcal{D} := \mathcal{D}_{\mathsf{t}}(S_1) \cup \cdots \cup \mathcal{D}_{\mathsf{t}}(S_k) \cup \{G\}$$

To conclude we only need to find a set of vertices S of G such that $\mathcal{D}_{\mathsf{t}}(S) = \mathcal{D}$. Let $S = S_1 \cup \cdots \cup S_k \cup \{\iota\}$, where ι is the interface of G. Let us show that $\mathcal{D}_{\mathsf{t}}(S) = \mathcal{D}$. This is a consequence of the following lemma:

▶ **Lemma 80.** *Let C be a context, K a graph and I an interface in H of the same arity as the hole of C. Suppose that the hole of C is not parallel to any module. We have:*

$$\mathsf{max\text{-}module}_{C[K]}(I) = \mathsf{max\text{-}module}_{K}(I)$$ ◀

▶ **Theorem 81.** *Suppose that $\mu x.L$ is a guarded iteration of type test. We have:*

$$L \textit{ is } \mathsf{CMSO} \textit{ definable} \quad \Rightarrow \quad \mu x.L \textit{ is } \mathsf{CMSO} \textit{ definable}$$

Proof. Let φ be a CMSO formula whose language is L. We transform the CMSO^d formula $\mu x.\varphi$ of Def. 75, whose language is $\mu x.L$, into a CMSO formula $\mu x^g.\varphi$ of the same language. The formula $\mu x^g.\varphi$ is obtained by replacing the quantification $\exists \mathcal{X}.$ by the set quantification $\exists S.$, and by replacing every sub-formula of $\mu x.\varphi$ of the form $(s, Z, t) \in \mathcal{X}$ by this formula:

$$(s = t) \ \wedge \ s \in S \ \wedge \ \text{“}(s, Z, t) \text{ is a maximal module”}$$

The last part of this formula is expressible in CMSO thanks to Prop. 32. The language of $\mu x^g.\varphi$ is the set of graphs for which we can find an L-decomposition encoded by a set of vertices S, and this is precisely the language $\mu x.L$ thanks to Prop. 79. ◀

5.2.2 The case of domain languages

Let $\mu x.L$ be a guarded iteration of type *domain*, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. Contrarily to the test case, the interfaces of $\mathcal{D}$ are not enough to reconstruct $\mathcal{D}$. Indeed, in this case, a component of $\mathcal{D}$ whose interface is v is not necessarily the maximal module at v, but some domain module of interface v, among possibly many others. A way to determine if a domain module is in the decomposition is to check whether it contains an interface of the decomposition. This works only for the components which are not the leaves of the decomposition. This is why we need to say explicitly which domain modules are the leaves. We do so by *coloring* the edges of the later.

In the following, we show that a set of vertices of a graph (representing the interfaces of a decomposition) together with a coloring of this graph (indicating which modules are leaves), is enough to recover the decomposition.

▶ **Definition 82** (Coloring, active modules.). *A* coloring *of a graph G is a set of its edges called* leaf edges. *A module of G is* active *if it contains a leaf edge.*

▶ **Definition 83** ($\mathcal{D}_{\mathsf{d}}(S, \mathsf{col})$). *Let G be a graph, S a set of vertices and* col *a coloring of G. We let $\mathcal{D}_{\mathsf{d}}(S, \mathsf{col})$ bet the set of active modules of G of type* d*, whose interfaces belong to S.*

▶ **Proposition 84.** *Let $\mu x.L : \mathsf{d}$ be a guarded iteration. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists S, \mathsf{col}. \quad \begin{array}{l} S \text{ is a set of vertices and } \mathsf{col} \text{ a coloring of } G \text{ such that} \\ \mathcal{D}_{\mathsf{d}}(S, \mathsf{col}) \text{ is an } L\text{-decomposition of } G. \end{array}$$

Proof. Similar to the proof of prop. 79. ◀

As in the previous section, we use Prop. 84 to get the following theorem.

▶ **Theorem 85.** *Suppose that $\mu x.L : \mathsf{d}$ be a guarded iteration. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

5.2.3 The case of parallel languages

The case of guarded iterations of type parallel is similar to the test case. Let $\mu x.L$ be a guarded iteration of type parallel, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. We show that the interfaces I of $\mathcal{D}$ is enough to recover the whole decomposition $\mathcal{D}$, because its components are the maximal modules of G whose interfaces belong to I. However, in this case, I is no longer a set of vertices, but a set of pairs of vertices, that is a relation on the vertices of G. We will show that this relation is necessarily a companion relation. Using this result and the fact that CMSO and CMSO^r have the same expressive power, we prove that the iteration is CMSO definable.

▶ **Definition 86** ($\mathcal{D}_{\mathsf{p}}(R)$). *Let G be a graph and R a relation on the vertices of G. We define $\mathcal{D}_{\mathsf{p}}(R)$ as the set of maximal modules of G, whose type is parallel, and whose interfaces belong to S.*

▶ **Proposition 87.** *Let $\mu x.L$ be a guarded iteration of type parallel. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists R. \quad \begin{array}{l} R \text{ is a set of vertices of } G \text{ and} \\ \mathcal{D}_{\mathsf{p}}(R) \text{ is an } L\text{-decomposition of } G. \end{array}$$

▶ **Proposition 88.** *Let $\mu x.L$ be an iteration of type parallel and let G a graph. The interfaces of every L-decomposition of G form a companion relation.*

Proof. We prove by induction on $n \geq 1$ that the interfaces of every L-decomposition of depth n of some graph G form a companion relation, witnessed by a set of paths P, such that the interface of G is witnessed by two parallel paths of P.

When $n = 1$, the decomposition $\mathcal{D}$ is reduced to the graph G. Since G is parallel, it has two parallel paths whose interface is the interface of G. Take P to be these two paths.

Suppose that $\mathcal{D}$ is a decomposition of depth $n + 1$. Hence it is of the form:

$$\mathcal{D} = \mathcal{D}_1 \cup \ldots \mathcal{D}_k \cup \{G\}$$

where $\mathcal{D}_i$ is an L-decomposition of depth at most n, of a graph G_i, for every $i \in [1,k]$. Let P_i be the set of paths provided by the induction hypothesis for $\mathcal{D}_i$, and let p_i, q_i be the two paths witnessing the interface of G_i, for $i \in [1,k]$.

We set $H := x\text{-}\mathsf{body}_{\mathcal{D}}(G)$. Since H is parallel, it has two parallel paths p and q whose interface is the interface of H. We transform the paths p and q of H into the paths p' and q' of G as follows. The paths p' and q' are obtained from p and q respectively by the following procedure: if e is an x-edge of H which is substituted by some G_i, then replace e by the path of p_i. Let P be the following set of paths:

$$P = (P_1 \setminus \{p_1\}) \cup \ldots (P_k \setminus \{p_k\}) \cup \{p', q'\}\,.$$

The set P is orthogonal and witnesses the interfaces of $\mathcal{D}$. Moreover, the interface of G is witnessed by two parallel paths of P, namely p' and q'. This concludes the proof. ◀

▶ **Theorem 89.** *Suppose that $\mu x.L$ is a guarded iteration of type parallel. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

5.2.4 The case of series languages

Let $\mu x.L$ be a guarded iteration of type series, G a graph in $\mu x.L$ and $\mathcal{D}$ an L-decomposition of G whose set of interfaces is I. As for the domain case, the set I is not enough to reconstruct the decomposition $\mathcal{D}$, and we need a coloring of the graph to determine which modules are the leaves of the decomposition $\mathcal{D}$. We show also that the set of interfaces I is a companion relation, which will be enough to conclude.

▶ **Definition 90** ($\mathcal{D}_{\mathsf{s}}(R, \mathsf{col})$)**.** *Let G be a graph, R a relation on the vertices of G and* col *a coloring of G. We let $\mathcal{D}_{\mathsf{s}}(R, \mathsf{col})$ bet the set of active modules of G of type series, whose interfaces belong to R.*

▶ **Proposition 91.** *Let $\mu x.L$ be a guarded iteration of type series. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists R, \mathsf{col}. \quad \begin{array}{l} R \text{ is a relation on the vertices of } G, \\ \mathsf{col} \text{ is a coloring of } G \text{ and} \\ \mathcal{D}_{\mathsf{s}}(R, \mathsf{col}) \text{ is an } L\text{-decomposition of } G. \end{array}$$

▶ **Proposition 92.** *Let $\mu x.L$ be a guarded iteration of type series and let G be a graph. The interfaces of every L-decomposition of G form a companion relation.*

▶ **Theorem 93.** *Suppose that $\mu x.L$ is a guarded iteration of type series. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

6 Recognizable implies regular

▶ **Theorem 94.** *If a language of tw_2-graphs is recognizable, then it is regular.*

We proceed gradually, by showing that this result holds for *domain-free graphs*, for *domain-free* graphs, then for tw_2 graphs.

▶ **Definition 95** (Domain-free)**.** *A graph is* domain-free *if all its domain modules are atomic.*

To give an example of how these proofs work, suppose that we have the following property:

▶ **Proposition 96.** *If a language of domain-free graphs is recognizable, then it is regular.*

Using Prop. 96, let us show the following property:

▶ **Proposition 97.** *If a language of domain graphs is recognizable, then it is regular.*

Proof. Let L be a language of domain graphs, $\mathcal{A}$ an algebra whose domain is D, $h : \mathbb{G}_{\mathsf{tw}_2}(\Sigma) \to \mathcal{A}$ a homomorphism and $F \subseteq D$ such that $h^{-1}(F) = L$. Let us show that L_v, the set of graphs over Σ whose image is v, is regular for every $v \in D$.

We associate every $v \in D$ with a new letter x_v and let $\Gamma := \{x_v \mid v \in D\}$. If $Q \subseteq D$, we denote by X_D the letters of Γ corresponding to these elements. We extend the homomorphism h to tw_2-graphs over the alphabet $\Sigma \cup \Gamma$ by letting $h(x_v) = v$ for every $x_v \in \Gamma$.

Let $v \in D$, $Q \subseteq D$ and $X \subseteq \Gamma$. We define the set of graphs $L_v^{Q,X}$ as follows. We let G be in this set if and only if:

- G is a domain graph over the alphabet $\Sigma \cup X$,
- the image of G is v,
- the image of the strict domain modules of G belong to Q.

Let us show that $L_v^{Q,X}$ is regular when $X_Q \cap X = \emptyset$. We proceed by induction on the size of Q. Suppose that $Q = \emptyset$. This case is based on the following lemma, obtained by case analysis on the graph G.

▶ **Lemma 98.** *Let G be a domain graph whose domain modules, distinct from G itself, are all atomic. There is a domain-free graph H such that $G = \mathsf{dom}(H)$.*

For every $w \in D$, let M_w^X be the set of domain-free graphs over the alphabet $\Sigma \cup X$ whose image is w. By Lem. 98, we have the following equation:

$$L_v^{\emptyset,X} = \bigcup_{\substack{w \in D \\ \mathsf{dom}(w) = v}} \mathsf{dom}(M_w)$$

which concludes the base case, thanks to Prop. 96. To handle the inductive case, we notice the following equality:

$$L_v^{Q \cup \{w\},X} = L_v^{Q,X \cup \{x_w\}}[\mu x_w.\ L_w^{Q,X \cup \{x_w\}}/x_w][L_w^{Q,X}/x_w] \qquad \blacktriangleleft$$

7 Conclusion

We are interested in studying the complexity-theoretic properties of our expressions. For instance understanding the complexity of deciding whether an expression is guarded, and what are the costs of translations between different formalisms (expressions, CMSO, algebra). This can help us get a better grasp of what role these expressions can play, and what is the fine interplay between these different formalisms. As stated in the introduction, this work on tree-width 2 graphs is meant to constitute a first step towards the case of tree-width k.

References

1 Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *LICS*, pages 407–416. ACM, 2016.

2 Mikolaj Bojanczyk and Michal Pilipczuk. Optimizing tree decompositions in MSO. In *STACS*, volume 66 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

3 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960. doi:0.1002/malq.19600060105.

4 Enric Cosme-Llópez and Damien Pous. K4-free graphs as a free algebra. In *MFCS*, volume 83 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

5 Bruno Courcelle and Joost Engelfriet. Graph structure and monadic second-order logic – A language-theoretic approach. In *Encyclopedia of mathematics and its applications*, 2012.

6 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1961.

7 Joost Engelfriet. A regular characterization of graph languages definable monadic second-order logic. *Theor. Comput. Sci.*, 88(1):139–150, October 1991. doi:10.1016/0304-3975(91)90078-G.

8 Zsolt Gazdag and Zoltán L. Németh. A kleene theorem for bisemigroup and binoid languages. *Int. J. Found. Comput. Sci.*, 22:427–446, 2011.

9 Ferenc Gécseg and Magnus Steinby. Tree languages. In *Handbook of Formal Languages*, 1997.

10 S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–40, 1956. Princeton.

11 Dietrich Kuske and Ingmar Meinecke. Construction of tree automata from regular expressions. *RAIRO – Theoretical Informatics and Applications – Informatique Théorique et Applications*, 45(3):347–370, 2011. doi:10.1051/ita/2011107.

12 Neil Robertson and Paul D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48:227–254, 1990.

13 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2:57–81, 2005.