

Separator logic and star-free expressions for graphs

Mikołaj Bojańczyk

Abstract. We describe two formalisms for defining graph languages, and prove that they are equivalent:

1. *Separator logic*.¹ This is first-order logic on graphs which is allowed to use the edge relation, and for every $n \in \{0, 1, \dots\}$ a relation of arity $n + 2$ which says that “vertex s can be connected to vertex t by a path that avoids vertices v_1, \dots, v_n ”.
2. *Star-free graph expressions*. These are expressions that describe graphs with distinguished vertices called ports, and which are built from finite languages via Boolean combinations and the operations on graphs with ports used to construct tree decompositions.

Furthermore, we prove a variant of Schützenberger’s theorem (about star-free languages being those recognized by a periodic monoids) for graphs of bounded pathwidth. A corollary is that, given k and a graph language represented by an MSO formula, one can decide if the language can be defined in either of two equivalent formalisms on graphs of pathwidth at most k .

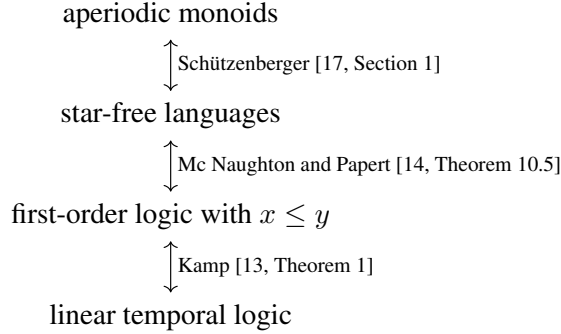
1. Introduction

In this introduction, we argue that although first-order definable languages of graphs, words and trees are intensively studied, there is a mismatch between the considered variants of first-order logic: **for graphs a local version that uses neighborhood only is typically studied, while for words and trees one typically studies a variant that has order and not just successor.** A consequence of this mismatch is that if we view a word as a special case of a graph, then the first-order definable word languages of words will not translate to first-order definable languages of graphs. **To overcome this mismatch, we propose a stronger variant of first-order logic for graphs, which can be seen as the natural graph generalisation of first-order logic with an order on positions.**

Let us begin with a discussion of the literature, starting with first-order logic for words and trees. In the field of logic and automata, when first-order logic is used to define a property of words, the vocabulary usually contains the order relation $x \leq y$ and not just the successor relation $x + 1 = y$. The difference is immaterial for monadic second-order logic MSO, but not for first-order logic, since order cannot be defined using first-order logic in terms of successor (successor can be defined in terms of order). Word languages that can be defined first-order logic with order have been intensively studied,

¹This logic was introduced independently in by Schrader, Siebertz and Vigny in [16]

starting in the 1960's, and are known to have many equivalent descriptions, which are summarized in the following diagram



Also for trees, first-order logic is usually used together with the descendant ordering, and not just successor (also called the child relation in the context of trees). Here, the appropriate temporal logic is CTL* [12, Main Theorem], and there is also a corresponding notion of star-free languages [3, Section 4]. This gives tree counterparts for three of the four descriptions for word languages. (The missing counterpart is “aperiodic monoids” – an algebraic characterization of first-order definable tree languages remains an major open problem [4, Problem 3].) Generally speaking, in this line of research the focus is on understanding the expressive power of the logic, ideally by giving an algorithm which inputs a regular language and decides if it can be defined in the logic. Logics for words and trees that have only the successor relation have also been studied, even if they might seem slightly less fundamental than the variants with order, and there exist algebraic and decidable characterizations, see [9, p.252] for the word case and [1, Theorem 1] for the tree case.

A different attitude is prevalent for graphs. Here, the usual notion of first-order logic uses the edge relation only, and has no predicates for reachability. As mentioned before, this difference is immaterial for monadic second-order logic. In first-order logic, however, reachability cannot be defined in terms of the edge relation. In the study of first-order logic on graphs, the focus is on finding efficient algorithms for model checking, with a famous milestone being that every sentence of first-order logic can be evaluated in almost linear time on every class of graphs that is nowhere dense [11, Theorem 1.1]. From a technical point of view, reasoning about first-order logic on graphs (with the edge relation only) usually relies on Gaifman locality, while reasoning about first-order logic on words and trees (with order) usually relies on compositionality.

As we can see from the above discussion, the traditional study of first-order logic has considered three cells in the following table:

	neighbour	reachability
words and trees	discussed	discussed
graphs	discussed	

The purpose of this paper is to fill the missing cell, by proposing a variant of first-order logic with reachability, which we call separator logic. We show that the class of languages definable in this logic

is reasonably robust, by presenting an equivalent notion of star-free expressions. The notions are designed so that if we view words and trees as a special case of graphs, then we recover the previously studied classes of first-order definable languages with order. Finally, we show an algebraic characterization, in the style of Schützenberger’s aperiodic monoids, of languages definable in separator logic for graphs of bounded pathwidth. **An extension of this characterization from bounded pathwidth to bounded treewidth seems to be beyond the reach of current methods, since it would require an algebraic characterization of first-order logic with descendant on trees.**

Acknowledgements. I would like to thank Nicole Schrader, Sebastian Siebertz and Alexandre Vigny for interesting discussions which happened after we learned that we had been independently studying the same logic. Also, I would also like to thank Colin Geniet, Michał Pilipczuk and Szymon Toruńczyk for their helpful remarks. Finally, I would like to acknowledge the financial support of the ERC Consolidator Grant *Lipa*, grant agreement 683080.

2. Separator logic and star-free languages

In this section, we describe the two formalisms used in this paper, namely separator logic and star-free languages of graphs, and we show that they are equivalent. Graphs are finite and undirected.

2.1. Separator logic

We begin our discussion with the logic, which is based on adding an infinite family of relations, apart from the edge relation, so that one can talk about separators in a graph. More formally, define its *separator model* of a graph to be the relational structure where the universe is the vertices, and which is equipped with the following relations:

$$\begin{array}{ll} \underbrace{E(v, w)}_{\text{there is an edge}} & \underbrace{S_n(v, w, u_1, \dots, u_n)}_{\text{for every } n \in \{0, 1, \dots\} \text{ there}} \\ \text{that connects } v \text{ and } w & \text{a relation } S_n \text{ of arity } n + 2 \text{ which} \\ & \text{says that } \{u_1, \dots, u_n\} \text{ is a} \\ & \text{separator between } v \text{ and } w \end{array}$$

The relations S_n are called *separator relations*. There are infinitely many separator relations in the model, and they have unbounded arity, but of course every formula uses finitely many relations. By convention, the relation $S_0(v, w)$ says that v and w are in different connected components of the graph. We use the name *separator logic* for first-order logic using the separator model. For example, the following formula says that a graph is disconnected:

$$\underbrace{\exists v \exists w S_0(v, w)}_{\text{there is a separator of size 0}}.$$

between some two vertices

Since $S_0(u, v)$ is the same as reachability, it follows that separator logic subsumes first-order logic with reachability. A corollary is that model checking for separator logic over planar graphs is as hard

as model checking for first-order logic on arbitrary graphs, i.e. it is XP complete. Here is another sentence, which says that the graph contains a cycle:

$$\underbrace{\exists v \exists w \forall u \neg S_1(v, w, u)}_{\text{there are two vertices that cannot be separated by one vertex}}.$$

In this paper, we are most interested in the expressive power of separator logic. An alternative research direction would be to search for efficient algorithms for model checking of separator logic on restricted graph classes. We do not study that research direction, beyond the following example, which was proposed by Michał Pilipczuk and Szymon Toruńczyk.

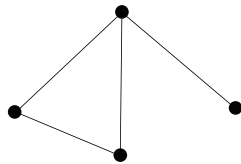
Example 2.1. (Complexity of model checking)

This example concerns the model checking problem for nowhere dense graph classes; in this example we assume that the reader is familiar with nowhere dense graph classes and the complexity of their model checking problem. For first-order logic with the edge relation only, the model checking problem (with the parameter being the formula) is fixed parameter tractable for every nowhere dense class, see [11, Theorem 1.1]. In contrast, the same model checking problem is not fixed parameter tractable for the class of all graphs, or more generally any graph class that is closed under subgraphs and not nowhere dense, subject to a standard assumption in the field of parameterized complexity, namely that $\text{AW}[*]$ is different from FPT.

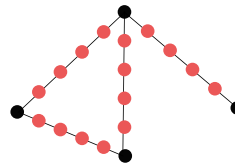
In this example, we show a graph class \mathcal{C} that is nowhere dense (in fact, it has the stronger property of bounded expansion), such that model checking of separator logic for this graph class is as hard as model checking of the usual first-order logic with the edge relation only over the class of all graphs. Therefore, the model checking problem for separator logic is unlikely to be fixed parameter tractable over this class. In other words, bounded expansion alone is not sufficient for tractability of the model checking problem for separator logic.

The idea is to simulate edges using long paths, thus making a graph sparse, and yet keeping enough information about the original graph so that it can be recovered using separator logic. For a graph, define its *subdivision* to be the result of subdividing each edge with n fresh vertices, where n is the number of vertices in the original graph, as explained in the following picture:

a graph with $n = 4$ vertices



its subdivision



For simplicity of presentation, the subdivision is seen as vertex coloured graph, with the vertices of the original graph being black, and the subdividing vertices being red. For such vertex coloured graphs, separator logic is extended with unary predicates for testing if a vertex has a given colour. The vertex colours could be eliminated by using suitable gadgets. Define \mathcal{C} to be the subdivisions – in the sense

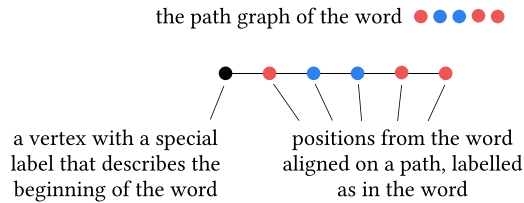
described above – of all graphs. We argue below that \mathcal{C} has bounded expansion, and model checking separator logic on \mathcal{C} is at least as hard as model checking first-order logic with the edge relation only on the class of all graphs.

1. We first prove that \mathcal{C} has bounded expansion. We assume that the reader is familiar with r -minors and bounded expansion. For $r \in \{0, 1, \dots\}$, define \mathcal{C}_r to be the class of depth- r minors of graphs from \mathcal{C} . To prove that \mathcal{C} has bounded expansion, we need to show that for every choice of r , graphs from \mathcal{C}_r have a number of edges that is at most linear in the number of vertices. To see this, one observes that with finitely many exceptions, every graph from \mathcal{C}_r has the property that every edge is adjacent to at least one vertex of degree at most two, and the latter property implies that the number of edges is at most twice the number of vertices.
2. We observe that the original graph, with its edge information, can be recovered from its subdivision using separator logic. The vertices are the black vertices. To check if two black vertices u_1 and u_2 are connected by a red path, we check if removing u_1 and u_2 gives a graph that has a nonempty purely red connected component; this can be expressed in separator logic.

The hardness result from this above example uses only the separator predicate S_2 , where the removed vertices are u_1 and u_2 . This is optimal, since first-order logic with the edge relation and the separator predicates S_0 and S_1 is no harder than first-order logic with edges only, as far as the model checking problem is concerned. For the predicate S_0 alone this is easy to see, since it is enough to compute the first-order theories of the connected components of a graph. A similar, if more complicated, argument can be made for S_1 , by computing the first-order theories of the 2-connected components.

Example 2.2. (Words as graphs)

In this example, we show that if words are viewed as graphs, then separator logic has the same expressive power as the usual notion of first-order logic for words that has the position order. Like in the previous example, to simplify notation we use vertex coloured graphs. For a word $w \in \Sigma^*$, define the *path graph* of w to be the vertex coloured graph, with the colours being Σ plus an extra black colour, that is described in the following picture:



The special black vertex on the left is used to orient the word. One could potentially avoid it by using directed edges; however it is unclear what is the right notion of separator logic for directed graphs. The left-to-right ordering on vertices of the path graph (with the special black vertex being the leftmost one) can be easily defined in separator logic: a non-black vertex u is to the left of a non-black vertex w if removing u separates w from the black vertex. Conversely, the separator predicates in a path graph

can be defined in first-order logic based on the order. This proves that a word language $L \subseteq \Sigma^*$ is definable in first-order logic with the position order if and only if the graph language

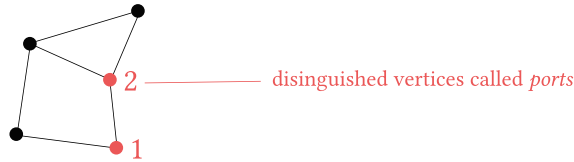
$$\{\text{path graph of } w : w \in L\}$$

is definable in separator logic. A similar argument also works for trees.

2.2. Star-free expressions for graphs

We now move to the second formalism considered in this paper, which is star-free expressions. These will have the same expressive power as separator logic.

These expressions are based on graphs with ports². Define a *graph with i ports* to be a graph with a tuple of i distinguished vertices, called *ports*. All ports must be pairwise different. Here is a picture of a graph with two ports:

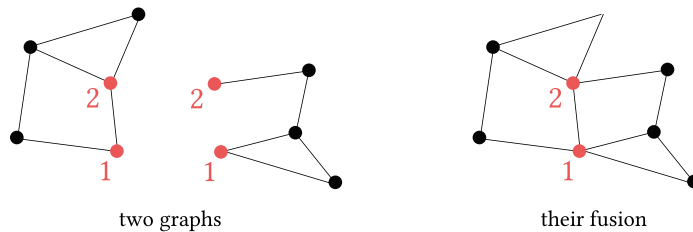


Since graphs with ports are the only kind of graphs that use, we call them graphs from now on. The *arity* of a graph is defined to be the number of ports. We use the following operations on graphs with ports.

Definition 2.3. (Treewidth operations)

The set of treewidth operations is the following (infinite) set of operations.

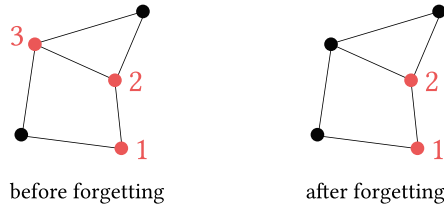
- **Fusion.** For every $k \in \{0, 1, \dots\}$ there is a k -fusion operation which inputs two graphs of arity k , and outputs the graph of arity k that is obtained by taking the disjoint union of the two input graphs, and then identifying, for every $i \in \{1, \dots, k\}$, the i -th port of both input graphs, as explained in the following picture:



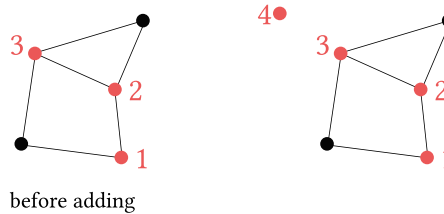
²Graphs with ports, and the operations on them, are based on Courcelle [10, Section 1.7]. Courcelle uses the word “source” instead of “port”. We use the name “treewidth operations” because a class of graphs has treewidth k if and only if it can be generated using the operations starting from graphs with at most edge, so that at most $k + 1$ ports are used at any given moment.

The two inputs to k -fusion might disagree on the subgraphs induced by the ports. As a result of fusion the edges are accumulated: if ports v and w are connected by an edge in at least one of two input graphs, then they are connected by an edge in the output graph.

- **Forget.** For every $k \in \{0, 1, \dots\}$ there is an operation which inputs a graph of arity $k + 1$, and outputs the graph of arity k that is obtained from the input graph by no longer distinguishing port $k + 1$.



- **Add.** For every $k \in \{0, 1, \dots\}$ there is an operation that inputs a graph of arity k , and outputs the graph of arity $k + 1$ that is obtained from the input graph by adding an extra isolated vertex, which becomes port $k + 1$.



- **Permute.** For every $k \in \{0, 1, \dots\}$ and every permutation of $\{1, \dots, k\}$, there is a unary operation on graphs of arity k , which reorders the list of ports according to the permutation.

Graphs together with the treewidth operations can be viewed as a multisorted algebra, with the arities being the sorts. Define a *graph language* to be a set of graphs, all of which have the same arity. The arity of the language is defined to be the arity of some (equivalently, every) graph in the language. When we take the complement of a graph language, we mean the complement with respect to all graphs of the given arity. (A corner case is the empty graph language: there is an empty language for every arity k . The complement of this graph language is the set of all graphs of arity k .) The treewidth operations can be applied to graph languages in the natural way, e.g. the k -fusion of two languages of arity k consists of all k -fusions where the first input is from the first language and the second input is from the second language. We now define the main object of this note, which is star-free languages of graphs.

Definition 2.4. (Star-free graph language)

The class of star-free graph languages is the least class of graph languages that contains all finite languages, and is closed under Boolean operations (including complementation) and the treewidth operations from Definition 2.3 .

In the above definition, the induction basis for the constructors is the finite languages. An alternative would be to use only the empty sets and the singleton sets for graphs with at most two vertices, since the remaining graphs can be constructed from such graphs using the treewidth operations. We finish this section with some examples of star-free graph languages.

Example 2.5. (Connected graphs)

We define the set of graphs of arity zero which are connected. We assume that, by definition, every graph is nonempty, i.e. it has at least one vertex (if we would allow the empty graph, then non-emptiness could be defined by taking a graph with one port, and forgetting that port). The set of all graphs of arity zero is therefore the complement of the empty set:

$$\neg \underbrace{(\emptyset : 0)}_{\text{the empty language of arity 0}}.$$

A graph of arity zero is disconnected if and only if it can be decomposed as the fusion of two nonempty graphs of arity zero. Therefore, the set of connected graphs of arity zero is defined by the star-free expression

$$\neg \underbrace{(\neg(\emptyset : 0) \oplus \neg(\emptyset : 0))}_{\text{disconnected graphs}}.$$

Example 2.6. (Contains an given induced subgraph)

Consider the language L of graphs of arity k which contain an edge from port i to port j . This language is star-free, because it can be described as the fusion of the set of all graphs of arity k , with the singleton language that contains exactly one graph – the graph of arity k where all vertices are ports and there is only one edge, namely from port i to port j . By taking a boolean combination of such graphs, we can use a star-free expression to specify which edges between ports are present, and which edges are not present. By forgetting all ports, we can define the language of graphs of arity zero which contain some fixed graph H as an induced subgraph.

The above example demonstrates how the ports can be used to simulate variables of logic, and how forgetting can be used to simulate existential quantification. This simple idea will be used to prove that all languages definable in separator logic are star-free.

Example 2.7. (Cycles)

A graph is a cycle if and only if it is connected, and every vertex has exactly two neighbors. This can be defined by a star-free expression.

Example 2.8. (Trees)

A graph is a tree if and only if it is connected, and every two non-neighboring vertices can be separated by some other vertex. Connectivity was already treated in the examples above. For the second property, consider the language L of graphs of arity 3, such that port 3 separates ports 1 and 2, i.e. every path from port 1 to port 2 must necessarily use port 3. A graph has this property if and only if it can be obtained by taking the 3-fusion of two graphs of arity 3, such that in the first graph port 2 is an

isolated vertex, and in the second graph port 3 is an isolated vertex. If we forget port 3 in the language L , then we get the language K graphs of arity 2 where the two ports are separated by some vertex. A connected graph of arity zero is a tree if for every way of selecting two non-neighboring ports, the resulting graph of arity 2 belongs to K .

2.3. Equivalence of the two formalisms

In this section, we prove that separator logic and star-free expressions define the same graph languages. To define a graph language of arity $k \in \{0, 1, \dots\}$ in separator logic, we use formulas with free variables: for a formula $\varphi(x_1, \dots, x_k)$ of separator logic with k free variables, its language is defined to be the set of graphs of arity k which satisfy the formula under the valuation which sets the i -th free variable to the i -th port. (This definition assumes an implicit order on the free variables.) Since the ports in a graph are required to be distinct, the graph language corresponding to a formula will only consider valuations where all free variables are different. For this reason, the language of the formula $x_1 = x_2$ is empty. We could have also considered a variant of graphs with ports which allow equalities on ports, with the same results, but we choose to work with the assumption that all ports are distinct.

Theorem 2.9. A graph language is star-free if and only if it is definable in separator logic.

The two implications in the theorem are proved in Section 2.3.1 and 2.3.2 below.

2.3.1. From separator logic to star-free expressions

We begin with the easier implication, which is from separator logic to star-free expressions. Since the ports in star-free expressions can be used to represent free variables, we can use a simple induction on formula size: we show that for every variables x_1, \dots, x_k and formula $\varphi(x_1, \dots, x_k)$ of separator logic, the corresponding language is star-free. (The formula does not need to use all variables. This happens for instance when taking a disjunction of two formulas that talk about different subsets of the variables.) For boolean combinations there is nothing to do, since star-free languages have boolean combinations built in. Consider now an existential quantifier

$$\exists x_{k+1} \varphi(x_1, \dots, x_k, x_{k+1}). \quad (1)$$

The rough idea is that existential quantification corresponds to forgetting the last port. There is one slightly subtle point here: in the definition of the language of a formula, we only consider valuations where all variables represent distinct vertices, since the definition of a graph with ports requires all ports to be distinct. Therefore, the language of the formula (1) consists of (a) the language of the formula φ with the forget operator applied to it; plus (b) the language of the formula

$$\bigvee_{i \in \{1, \dots, k\}} \varphi(x_1, \dots, x_k, x_i).$$

The languages used in (a) and (b) are star-free thanks to the induction assumption, thus proving the induction step.

We are left with the induction basis, which corresponds to the edge and separator predicates in the logic. Consider first the edge relation. The corresponding language is the graphs of arity k where some two ports $i, j \in \{1, \dots, k\}$ are connected by an edge; this language is star-free as we have shown in Example 2.6. Consider now the the separator predicate. The corresponding graph language consists of graphs of arity k , such that some two ports $s, t \in \{1, \dots, k\}$ are separated by a subset of ports $I \subseteq \{1, \dots, k\}$. We need to justify that this language is star-free. It is not hard to see that a graph belongs to this language if and only if

- (*) there exists a partition of $\{1, \dots, k\} - I$ into disjoint sets I_1, \dots, I_ℓ and the graph can be decomposed as a k -fusion

$$G_1 \oplus \dots \oplus G_\ell$$

so that s and t are in different blocks of the partition, and for every $i \in \{1, \dots, \ell\}$ all of the ports from outside $I \cup I_i$ are isolated in G_i .

The idea behind (*) is that two ports are in the same block of the partition if they are equal or they can be connected by a path that avoids ports with indices in I . Finally, condition (*) is easily seen to be definable, because the partition can be chosen in finitely many ways, and because “port i is an isolated vertex” is a star-free language.

2.3.2. From star-free expressions to separator logic

By induction on the size of a star-free expression, we show that corresponding language can be defined in separator logic. The interesting case is the k -fusion operation. By induction assumption, we know that both fused languages can be defined in separator logic.

The difficulty is that the definition of fusion cannot be directly formalized in separator logic, since this would require saying that one can partition the non-port vertices into two parts which induce graphs in the two fused languages. On the face of it, separator logic does not have a mechanism to quantify over such partitions. However, we can work around this difficulty using a compositionality argument. The key idea for this proof is that the type of a graph (which is the information about the graph with respect to separator logic of given quantifier rank) can be inferred from the types of its prime factors, which are connected components of the graphs after the ports have been removed. As we will also see, the types of the prime factors will also give us enough information to decide if a graph belongs to the fusion of two languages definable in separator logic. A more detailed argument is presented below.

We use the usual notion of quantifier rank. For $r \in \{0, 1, \dots\}$, we say that two graphs are r -equivalent if they have the same arity $k \in \{0, 1, \dots\}$ and they belong to the same languages defined by formulas of separator logic that have quantifier rank at most r .

Claim 2.10. (Congruence)

Let $r, k \in \{0, 1, \dots\}$, and let \equiv be r -equivalence on graphs of arity k . Then \equiv is a congruence with respect to k -fusion, i.e.

$$\bigwedge_{i=1,2} G_i \equiv G'_i \quad \Rightarrow \quad G_1 \oplus G_2 \equiv G'_1 \oplus G'_2$$

holds for all graphs G_1, G_2, G'_1, G'_2 of arity k .

Proof:

A standard Ehrenfeucht-Fraïssé pebble game argument. We assume that the reader is familiar with such arguments, and hence we only sketch it. For $i \in \{1, 2\}$, define the i -th *local game* to be the Ehrenfeucht-Fraïssé game corresponding to the equivalence of G_i and G'_i , and define the *composite game* to be the game corresponding to the equivalence of $G_1 \oplus G_2$ and $G'_1 \oplus G'_2$. Using a standard composition of strategies, Duplicator uses a strategy in the composite game which has the property that whenever a position in the composite game is reached, then for every $i \in \{1, 2\}$, if we keep only the vertices and pebbles from G_i and G'_i , then we get a position in the i -th local game that is consistent with Duplicator's winning strategy. We will now prove that this composite strategy is winning in the composite game. To prove this, we need to show that if all r rounds have been played, thus reaching two pebbblings

$$\underbrace{x_1, \dots, x_r}_{\text{a pebbling in } G_1 \oplus G_2} \quad \text{and} \quad \underbrace{x'_1, \dots, x'_r}_{\text{a pebbling in } G'_1 \oplus G'_2} \quad (2)$$

then the same quantifier-free formulas are satisfied on both sides. The interesting case is that of a quantifier-free formula which is a separator relation. To show that the same separator relations, we need to show that if, on one side of the equivalence, the i -th pebble can be connected with the j -th pebble by a path that avoids pebbles from a set $I \subseteq \{1, \dots, r\}$ for some set I of at most ℓ elements, then the same is true on the other side of the equivalence. Suppose that such a connecting path exists on one side of the equivalence, say the left side. To show that a similar connecting path exists on the other side, we decompose the path on the left side into segments, such that in each segment the ports are not used except for the source and target vertices. By definition of k -fusion, each such segment is necessarily contained in either G_1 or G_2 , and therefore a corresponding segment can be found in G'_1 or G'_2 . The corresponding segments form a path in $G'_1 \oplus G'_2$ which connects the i -th pebble to the j -th without passing through pebbles from I . \square

Let us now prove that languages definable in separator logic are closed under k -fusion. Suppose that L_1 and L_2 are languages of arity k that are definable in separator logic. Choose the quantifier rank $r \in \{1, 2, \dots\}$ so that both languages L_1 and L_2 are defined by formulas of separator logic which have quantifier rank at most r . Define the r -type of a graph of arity k to be its equivalence class with respect to r -equivalence. By choice of r , for every $i \in \{1, 2\}$, membership of a graph in L_i depends only on its r -type.

A formula of separator which k free variables and quantifier rank r will have $k + r$ variables, and therefore it can use only the separator predicates of arity at most $k + r$. In particular, up to logical equivalence there are finitely many equivalence classes, once the arity k and the quantifier rank r have been fixed. This means that there are finitely many r -types of graphs of arity k .

We can view graphs of arity k as a commutative monoid, where the monoid operation is k -fusion. By the congruence property from Claim 2.10, having the same r -type is a monoid congruence, and hence the r -types form a finite commutative monoid. Also, the monoid of r -types is aperiodic, in the sense that there is some threshold $m \in \{0, 1, 2, \dots\}$ such that for every a in this monoid, taking the

fusion of a with itself m times gives the same result as taking the fusion of a with itself $m + 1$ times. In fact, aperiodicity holds not only for separator logic, but even for monadic second-order logic [5, Lemma 4.19].

For a graph G , define a *prime factor* of G to be any graph H that is obtained from G by taking some non-port vertex v , and taking the subgraph of G that is induced by the ports and those vertices which can be reached from v by a path that does not visit ports. Every graph is a fusion of its prime factors

$$G = \underbrace{G_1 \oplus \cdots \oplus G_n}_{\text{here } \oplus \text{ denotes fusion on graphs of arity } k}.$$

From the observations about congruence described above, it follows that for every graph of arity k , its r -type is uniquely determined by the following information: for every type a , how many prime factors of G have r -type a , counted up to threshold m from the definition of aperiodicity. We use the name m -profile for this information.

The multiset of prime factors of $G_1 \oplus G_2$ is the multiset union of the prime factors of G_1 and G_2 , and therefore

$$G \in L_1 \oplus L_2$$

if and only if the multiset of prime factors in G can be partitioned into two multisets, the first multiset having an m -profile that is consistent with L_1 , and the second multiset having an m -profile that is consistent with L_2 . Whether or not this is possible can be deduced from the $2m$ -profile of the original graph. Finally, the $2m$ -profile can be defined in separator logic. This completes the proof that languages definable in separator logic are closed under fusion.

3. Bounded pathwidth and aperiodicity

In this part of the paper, we try to develop an algebraic theory for star-free languages of graphs, in the case of bounded pathwidth. Our goal is to give a version of Schützenberger’s theorem about star-free languages and aperiodic monoids, see [17, Section 1]. This theorem says that a word language is star-free if and only if it is recognized by a finite aperiodic monoid, i.e. a monoid M such that for every $a \in M$ there is some power $m \in \{0, 1, \dots\}$ such that

$$a^m = a^{m+1}.$$

Equivalently, a finite monoid is aperiodic if and only if it only contains trivial groups (a group contained in a monoid is any subset of the monoid such that the monoid multiplication induces a group structure, with a group identity that is not necessarily the same as in the monoid).

In this section, we prove a similar theorem for graphs of bounded pathwidth, which characterizes the star-free languages (equivalently, those definable in first-order logic) by a certain aperiodicity condition. A corollary of our result is that there is an algorithm, which inputs a language of graphs of bounded pathwidth (e.g. given by a formula of monadic second-order logic), and decides whether

or not the language is star-free (equivalently, definable in separator logic). The usefulness of such an algorithm is perhaps debatable; but the main point of the characterization is that it can be seen as further evidence of the idea that our notion of star-free languages for graphs is consistent with the spirit of star-free languages for finite words.

The results of this section concern bounded pathwidth. Ideally, we would want an algebraic characterization that works for more graphs, e.g. for graphs of bounded treewidth. Such a characterization seems to be beyond the reach of our techniques. This is because already for trees, i.e. for graphs of treewidth 1, finding an algebraic characterization of tree languages definable in first-order logic with descendant is a major open problem [4, Problem 3].

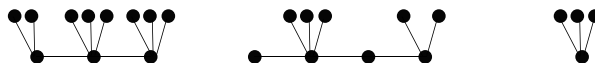
Monoids. Our algebraic approach to graphs of bounded pathwidth is based on [7, Section 4.2.1]: we view path decompositions as words over an alphabet that describes graph operations, and to recognize properties of path decompositions we use homomorphisms from such words into finite monoids.

We begin with a quick summary of monoids and their properties. Recall that a *monoid* is a set equipped with a binary associative operation, denoted by $a \cdot b$, together with a neutral element 1 which satisfies $1 \cdot a = a = a \cdot 1$. A *monoid homomorphism* is a function between two monoids that commutes with the monoid operation and preserves the neutral element. We say that a monoid homomorphism $\alpha : A \rightarrow B$ recognizes a subset $L \subseteq A$ if elements of A with equal value under α have equal membership status for L . We say that a monoid homomorphism $\alpha_1 : A \rightarrow B_1$ *refines* a homomorphism $\alpha_2 : A \rightarrow B_2$ if equal values for α_1 imply equal values for α_2 ; this is the same as saying that α_2 factors through α_1 . Every subset L of a monoid A (we use the name *language* for such subsets) is known to have a *syntactic homomorphism*, i.e. a homomorphism which recognizes the language, and which is refined by every other surjective homomorphism that recognises the language. One usually cares about languages contained in monoids that are finitely generated and free; but in this paper will be interested in monoids that represent graphs of bounded pathwidth, and such monoids will not be free.

3.1. Pathwidth and contexts

Consider a graph (for the moment, without any ports). A *path decomposition* of this graph is defined to be a sequence of subsets of vertices, called *bags*, such that: (1) every vertex appears in some bag and for every edge there is a bag that contains both endpoints; and (2) for every vertex, the bags which contain this vertex form an interval (i.e. if a vertex appears in two bags from the sequence, then it appears in all other bags between these two). The width of a path decomposition is the maximal bag size, minus one. The *pathwidth* of a graph is the minimal width of its path decompositions.

Example 3.1. A graph has pathwidth 1 if and only if it can be obtained from a disjoint union of paths by replacing each vertex by a star, as in the following picture:



This property can be defined in separator logic. For higher pathwidth the situation is harder, and we believe that separator logic cannot define pathwidth k for all but finitely many k .

For pathwidth, the appropriate notion of graphs with ports is *contexts*, which were called bi-interface graphs in [7, Definition 4.5]. These are like graphs with ports, except that there are two kinds of ports, called left and right ports.

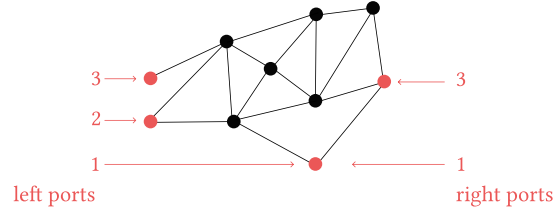
Definition 3.2. (Context)

A *context* of arity $k \in \{0, 1, \dots\}$ is defined to be a graph together with two partial injective maps

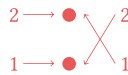
$$\text{left, right} : \{1, \dots, k\} \rightarrow \text{vertices of the graph.}$$

For $i \in \{1, \dots, k\}$, the i -th *left port* of the context is defined to be the image of i under the left map; such a vertex is said to have left index equal to i . Likewise for right ports. We require the following compatibility of left and right indices: if both the left and right indices of a vertex are defined, then these indices are equal.

Here is a picture of a context of arity 3, where the 2-nd right port is undefined, and the 1-st left port is equal to the 1-st right port:

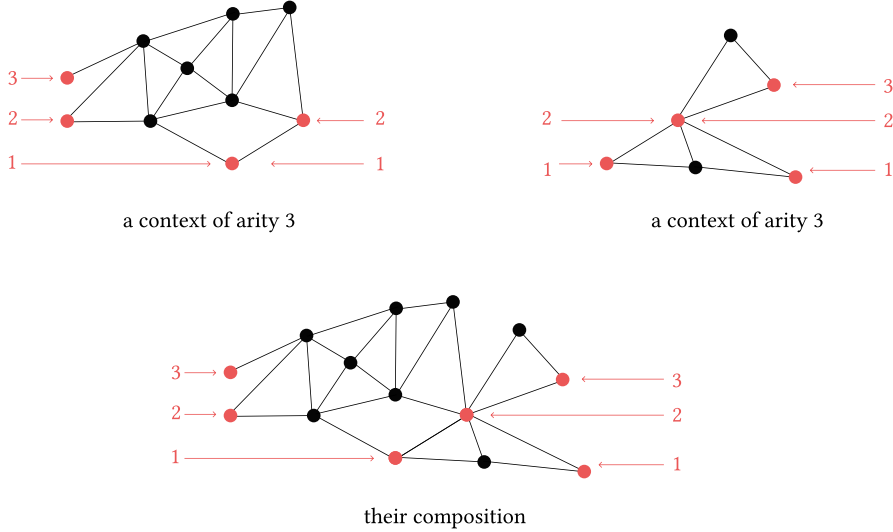


A vertex which is both a left and right port is called a *persistent port*, such ports will play an important role later in the paper. The compatibility of indices in the definition of contexts is meant to disallow contexts such as this one



which would cause problems for star-free expressions.

The idea behind contexts is that they describe path decompositions, with the left ports being the contents of the leftmost bag, and the right ports being the contexts of the rightmost bag. Composition of contexts, which corresponds to concatenation of path decompositions, is defined in the same way as fusion of graphs with ports, and is illustrated in the following picture:

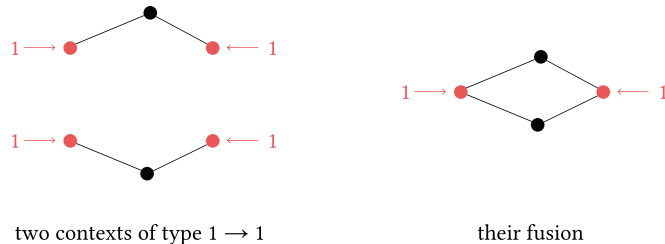


Contexts arity k form a monoid, which is denoted by P_k . The identity of this monoid is the context where all ports are defined and persistent, and there are no edges and no other vertices. A graph without ports can be viewed as a context of any arity k , where all ports are undefined. We say that context has pathwidth at most k if the underlying graph has a path decomposition of width at most k , where the first bag contains all the left ports and the last bag contains all the right ports. The following lemma is a straightforward reformulation of the definition of pathwidth; with the generating context playing the role of bags in a path decomposition.

Lemma 3.3. [7, Lemma 4.6] A context has pathwidth at most k if and only if it can be generated, in the semigroup P_k , by contexts which have at most $k + 1$ vertices.

We will use the name k -generators for the generating contexts in the above lemma, i.e. contexts of arity k with at most $k + 1$ vertices.

Remark 3.4. In our algebraic approach to bounded pathwidth, we allow only one operation on contexts, namely composition. In the terminology of series parallel graphs; context composition corresponds to serial composition of graphs. A parallel composition of contexts can also be defined, call it *fusion of contexts*, as illustrated in the following picture:



Unfortunately, we cannot add fusion to our algebras, because contexts of given pathwidth are not closed under fusion. For example, in the picture above the two fused contexts have pathwidth 1, but their fusion has pathwidth 2. In fact, adding fusion to the monoid of pathwidth k would allow us to generate all graphs of treewidth k , since we can view a graph with k ports as a context where only the left k ports are defined, and the right ports are all undefined. Using fusion on such contexts would allow us to simulate arbitrary tree decompositions of width k .

3.1.1. Recognizable languages of pathwidth k

We will be interested in classifying recognizable languages of given pathwidth, i.e. languages which are recognized by a monoid homomorphism from the monoid P_k to a finite monoid. Let us begin with an important example of a homomorphism, which is called here the *reachability homomorphism* but was called *abstraction* in [7, Section 4.2.1].

Example 3.5. (Reachability homomorphism)

An *inner path* in a context is defined to be a path that does not use ports, with the possible exception of its source and target vertices. Define the *reachability homomorphism*, for contexts of arity $k \in \{1, 2, \dots\}$, to be the function which maps a context of arity k to the following information: (a) which indices $i \in \{1, \dots, k\}$ describe persistent ports; and (b) for which pairs

$$(x, y) \in \{\text{left}, \text{right}\} \times \{1, \dots, k\}$$

is there an inner path from port x to port y . The reachability homomorphism is indeed a homomorphism, i.e. if we know the values of this homomorphism for two contexts w and v , then we also know the value of this homomorphism for the composition $w \cdot v$. An example of a language that is recognized by the reachability homomorphism is the contexts which admit an inner path from left port 1 to right port 1.

We can describe properties of contexts using logic, e.g. separator logic, by associating a model to each context. The ports of the context are described using $2k$ constants, with k constants for the left ports and k constants for the right ports. Since ports need not be defined in a context, we use a slightly non-standard version of first-order logic where constants can be undefined; to check if a constant is defined we use the convention that $c = c$ gives a “no” answer for constants c that are undefined. Contexts of bounded pathwidth admit a version of Courcelle’s Theorem: if φ is a sentence of monadic second-order logic (possibly with modulo counting quantifiers), then for every $k \in \{1, 2, \dots\}$ there is a homomorphism

$$\alpha : P_k \rightarrow A$$

into a finite monoid which recognises φ on contexts of pathwidth at most k . The proof is the same as in Courcelle’s Theorem. The situation is more complicated for the converse of Courcelle’s Theorem – a formal proof of the converse of Courcelle’s Theorem for contexts of bounded pathwidth would require a nontrivial reworking of the existing proof for treewidth, see [7, Theorem 2.10], and we do not do this here³.

³The issue is that the transductions in [7] output tree decompositions, even if the input graphs have bounded pathwidth. This

3.2. The characterization

We now move to the main subject of Section 3, which is an algebraic characterization of those languages of pathwidth at most k which are star-free, or equivalently, definable in separator logic. We begin with the following lemma, which establishes what it exactly that we mean when talking about star-free languages of bounded pathwidth, at least for graphs without ports.

Lemma 3.6. Let L be a set of graphs without ports. The following conditions are equivalent for every $k \in \{1, 2, \dots\}$:

1. L is subset of the monoid P_k that can be generated from finite sets, by using the monoid operation (context composition) and Boolean operations (with complementation relative to P_k);
2. L is the intersection of P_k with some language that is star-free in the sense of Definition 2.4.

Proof:

A straightforward induction on the size of expressions. □

The lemma speaks about graphs without ports, because these can be viewed both as contexts (as in item 1 of the lemma), and as graphs with ports (as in item 2 of the lemma). From now on, we will be more interested in the context perspective from item 1, and we will use the definition from item 1 when talking about star-free subsets of P_k that use contexts which have defined left and right ports.

Remark 3.7. Note that it might be the case that a language is star-free as subset of P_k , but not as a subset of P_{k+1} . A natural candidate for this phenomenon is P_k itself, which seems unlikely to be a star-free as a subset of P_{k+1} , as we have already mentioned in Example 3.1. We do not actually check whether this candidate is correct, but such a check could be performed automatically: later on in this section we will present a result, see Corollary 3.11, which implies that there is an algorithm that inputs $k \in \{1, 2, \dots\}$ and answers whether or not P_k is a star-free subset of P_{k+1} .

The issues from the above remark also explain why we do not consider a third potential notion of star-free of pathwidth at most k : languages which are star-free according to Definition 2.4, and which simultaneously happen to be contained in P_k . In this third notion, it would be the job of the star-free expression to ensure that the graph has low pathwidth.

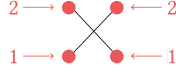
The main goal of Section 3 is to generalize Schützenberger’s characterization of star-free languages, from words to graphs of bounded pathwidth. The rough idea is that these are the languages that can be recognized by homomorphisms into finite aperiodic monoids. This idea is not completely correct, because if we use the original definition of aperiodic monoid, then there are languages which are star-free languages and yet periodic, as shown in the following example.

Example 3.8. (Periods from reachability)

Let $L \subseteq P_2$ be the set of contexts of arity 2 that admit a path from left port 1 to right port 1. This language is easily seen to be definable in separator logic, and therefore it is also star-free. However,

seems necessary, since defining a path decomposition would require imposing something similar to a linear order on the vertices, and this is impossible for certain simple graphs, e.g. graphs without edges.

the syntactic monoid of this language contains a copy of the two-element group. Indeed, let w be the following context:



If we consider powers w^n of this context, then the syntactic homomorphism of L will assign different values to even and odd powers, because left port 1 can be connected to right port 1 in w^n if and only if n is even.

As the above example shows, our algebraic characterization of star-free languages cannot simply say that the recognizing monoid is aperiodic. However, it turns out that the periods described in the above example are the only kind of periods that are permitted for star-free languages, as formalized in the following theorem, which is the main contribution of the present Section 3.

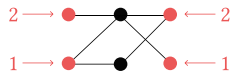
Theorem 3.9. A language $L \subseteq P_k$ is star-free if and only if it is recognized by a homomorphism $\alpha : P_k \rightarrow A$ into a finite monoid A such that every context $w \in P_k$ satisfies the following implication:

$$\underbrace{\beta(w) \text{ is idempotent}}_{\substack{\text{here } \beta \text{ is the} \\ \text{reachability homomorphism} \\ \text{from Example 3.8}}} \Rightarrow \underbrace{\alpha(w) \text{ is aperiodic}}_{\substack{\text{an element } a \in A \text{ is called } \textit{aperiodic} \text{ if} \\ a^m = a^{m+1} \text{ holds for some } m \in \{1, 2, \dots\}}}$$

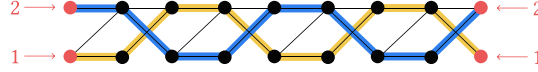
A homomorphism that satisfies the implication in the above theorem is called *aperiodic modulo reachability*. One could also imagine variants of the definition above, with other homomorphisms than the reachability homomorphism used for β ; maybe such variants would correspond to variants of separator logic, e.g. the disjoint paths logic from [16, Section 4].

One application of the theorem is to prove that certain languages are not star free, as in the following example, which corresponds to [16, Corollary 3.10].

Example 3.10. Let $L \subseteq P_2$ be the contexts with the following property: there exist two vertex disjoint paths, one from left port 1 to right port 1, and another one from left port 2 to right port 2. We will show that this language is not star-free, because its syntactic homomorphism is not aperiodic modulo reachability. (If any recognizing homomorphism is aperiodic modulo reachability, then the syntactic homomorphism is as well.) Indeed, let w be the following context



This context is mapped to an idempotent by the reachability homomorphism: all powers of this context have the property that all ports are distinct, and every two ports can be connected by an inner path. However, we will show that the image of this context under the syntactic homomorphism of the language L is not aperiodic, and therefore the language is not star-free. Indeed, if m is even, then the context w^m belongs to the language, as explained with two paths (blue and orange) in the following picture for $m = 4$:



On the other hand, if m is odd, then it is impossible to find two such paths. This example shows that separator logic cannot express the existence of two vertex disjoint paths with given sources and targets.

The characterization in Theorem 3.9 is effective, as expressed in Corollary 3.11 below. In the corollary, we use monadic second-order logic to define properties of contexts. This is the MSO_1 logic of Courcelle: it can quantify over vertices, sets of vertices, and it is equipped with a binary edge relation.

Corollary 3.11. Given $k \in \{1, 2, \dots\}$ and a formula φ of monadic second-order logic, one can decide if there is a formula of separator logic that is equivalent to φ on graphs of pathwidth at most k .

Proof:

We assume that the formula describes a property of graphs without ports, but the same proof would also work with ports. By Courcelle's Theorem, one can compute a monoid homomorphism

$$\alpha : P_k \rightarrow A$$

into a finite monoid which recognizes the language of φ on graphs of pathwidth at most k . Next, using a natural fixpoint procedure (called Moore's minimization algorithm), one can minimize the monoid, and thus we can assume that h is the syntactic homomorphism. As mentioned in Example 3.10, if any recognizing homomorphism is aperiodic modulo reachability, then the syntactic homomorphism has this property as well. Therefore, by Theorems 2.9 and 3.9, the formula φ is equivalent to a formula of separator logic on graphs of pathwidth at most k if and only if the syntactic homomorphism α is aperiodic modulo reachability. The latter property can be checked effectively, by an exhaustive search of the product of the syntactic monoid A with the target monoid of the reachability homomorphism. \square

In the above corollary, the language is represented by a formula of monadic second-order logic (one could also use the extension of this logic which allows modulo counting). From the point of view of computational complexity, a bottleneck in the algorithm is Courcelle's Theorem, where the conversion from logic to a monoid requires a tower of exponentials. This tower can be avoided if the language is represented by a homomorphism into a finite monoid; in this case the algorithm runs in polynomial time.

The rest of Section 3 is devoted to proving Theorem 3.9.

The easier implication in the theorem is that if a language is star-free, then it is recognized by a homomorphism that is aperiodic modulo reachability. For this proof we use separator logic, and a standard Ehrenfeucht-Fraïssé proof, which is only sketched. For a quantifier rank $r \in \{0, 1, \dots\}$ define the r -type of a context w of arity k to be the set of sentences of separator logic that have quantifier rank at most r and are true in the context (as mentioned before, we assume that the logic

is equipped with $2k$ constants for the left and right ports). A standard compositionality argument, similar to Claim 2.10, shows that for every r , the function which maps a context to its r -type is a homomorphism into a finite monoid. Furthermore, if r is the quantifier rank of the formula that defines L , then this homomorphism recognizes the language L . The following lemma shows that this homomorphism is aperiodic modulo reachability, thus proving the easier implication in Theorem 3.9.

Lemma 3.12. Suppose that w is a context which is mapped to an idempotent by the reachability homomorphism. For every quantifier rank r , there is some $m \in \{1, 2, \dots\}$ such that the contexts w^m and w^{m+1} have the same r -type.

Proof:

Take m to be 3^r . A vertex of w^m has a representation as a pair (vertex of w , number in $\{1, \dots, m\}$), with the number indicating which copy of w inside w^m is used. This representation is not unique, due to ports being shared across consecutive copies. To ensure uniqueness, define the *canonical representation* to be the representation which uses the smallest number on the second coordinate. This canonical representation is unique, and its two coordinates are called the *offset* (a vertex of w) and *index* (a number in $\{1, \dots, m\}$), respectively. Similarly we define the index and offset for vertices of w^{m+1} . Using a standard argument, see [18, proof of Theorem IV.2.1], one can show that Duplicator has a strategy in the r -round Ehrenfeucht-Fraïssé game on w^m and w^{m+1} which ensures that once all r rounds have been played and $i, j \in \{1, \dots, r\}$ are pebble names, then: (1) the offsets of pebble i in w^m and w^{m+1} are the same; and (2) the order on indices of pebbles i and j are the same in w^m and w^{m+1} ; and (3) the indices of pebbles i and j differ by a number that is not 1, both in w^m and w^{m+1} . Because of the above assumptions, and since one or more copies of w are equivalent with respect to the reachability homomorphism, it follows that the two pebbleings satisfy the same reachability predicates. This, in turn, means that Duplicator's strategy was winning. \square

This completes the proof of the easier implication in Theorem 3.9.

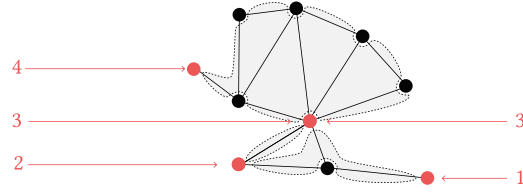
3.3. From aperiodicity to separator logic

The rest of Section 3 is devoted to the harder implication of Theorem 3.9, which says that if a language is recognized by some homomorphism that is aperiodic modulo reachability, then the language is star-free. When proving the harder implication, we use the same strategy as Schützenberger's original proof for words. There are, however, extra arguments that are specific to graphs. These arguments revolve mainly around the notion of bridges, which is described below.

3.3.1. Bridges

We partition the edges of a context into *inner components* as follows: two edges in a context are in the same inner component if they can be connected by using non-port vertices and other edges. In other words, two edges are in the same inner component if there is some inner path that uses both of the edges.

Example 3.13. Here is a context with its inner components:



a context with its edges partitioned into inner components

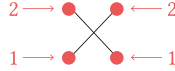
We say that an inner component is *incident* to some port if some edge from the inner component is incident to that port.

Definition 3.14. (Bridge)

An inner component in a context is called a *bridge* if it is incident to a left non-persistent port and also incident to a right non-persistent port. (Recall that persistent ports are those which are both left and right ports.)

Example 3.15. In the context from Example 3.13, there is exactly one bridge, namely the inner component at the bottom of the picture which is incident to left port 2 and right port 1.

Example 3.16. Consider the problematic context



from Example 3.8, which is periodic with respect to the reachability homomorphism. In this context, there are no persistent vertices, and two bridges, namely each of the two edges is a singleton bridge. Hence, this problematic context has two bridges.

In the Example 3.16, two bridges were used to get a period under the reachability homomorphism. It turns out that this is the case in general.

Lemma 3.17. If a context has at most one bridge, then its image under the reachability homomorphism is aperiodic.

Proof:

Let w be a context of arity k . The reachability homomorphism tells us which ports are defined, which ports are persistent, and which pairs of source and target ports can be connected by an inner path. The first two pieces of information, about which ports are defined and persistent, does not change as we take powers w^n of a context. The interesting part is the connectivity using inner paths.

Suppose that we want to know if two ports are connected by an inner path in a power w^n . Suppose first that the two ports are on the same side, say both are left ports. This case is dealt with in the following claim.

Claim 3.18. Let $s, t \in \{1, \dots, k\}$. All but finitely many powers $n \in \{1, 2, \dots\}$ lead to the same answer to the question: does w^n have an inner path from the s -th left port to the t -th left port?

Proof:

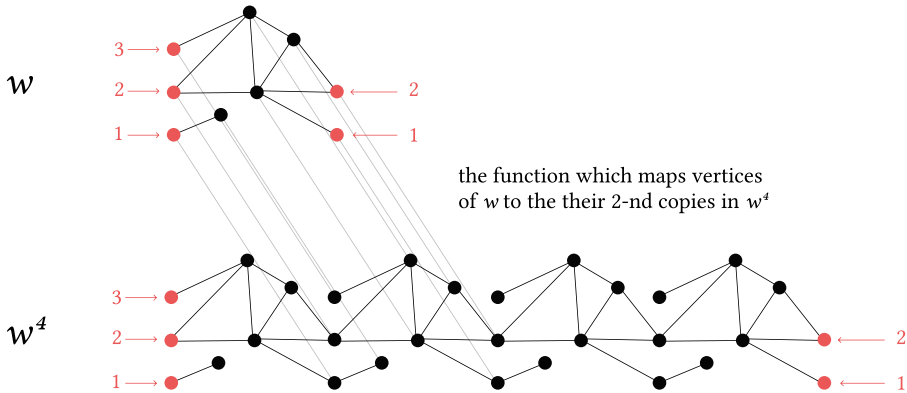
It is easy to see that if there is an inner path from the s -th left port to the t -th left port in w^n , then the same path is also an inner path in w^{n+1} (here it is important that both the source and target of the inner path are on the same side of the context). Therefore, the answer to the question in the claim can only go from “no” to “yes”, and so it eventually stabilizes. \square

A symmetric claim holds when both the source and target are right ports. We are left with the case when the source is a left port and the target is a right port. Also, none of these ports are persistent, since otherwise they would both be left ports, or they would both be right ports, and we could use Claim 3.18. This remaining case is dealt with by the following claim, which completes the proof of the lemma.

Claim 3.19. Let $s, t \in \{1, \dots, \ell\}$ be such that the s -th left port and the t -th right port are both non-persistent. All but finitely many $n \in \{1, 2, \dots\}$ lead to the same answer to the question: does w^n have an inner path from the s -th left port to the t -th right port?

Proof:

For numbers $n \in \{1, 2, \dots\}, i \in \{1, \dots, n\}$, we define the i -th copy of a vertex of w inside w^n as explained in the following picture:



These copies are not disjoint, since port vertices appear in several copies. In the proof of this claim, we assume without loss of generality that w does not have any persistent ports. The above picture does not use persistent ports, and indeed this can be assumed without loss of generality (which will simply the reasoning). Indeed, the answer to the question in the claim does not change if replace w^n by either of the following two contexts, which are furthermore equal to each other: the context obtained from w^n by removing persistent ports; or the n -th power of the context obtained from w by removing persistent ports. From now on, we assume that w has no persistent ports.

Define S_i^n to be the vertices x of w such that w^n admits an inner path from the s -th left port to the i -th copy of x . We will use the assumption on having a unique bridge to show that

$$S_i^n \subseteq S_{i-1}^n. \quad (3)$$

To prove the above inclusion, consider an inner path π in w^n that begins in the s -th left port and reaches the i -th copy of x for $i > 2$. The key observation is that, since w has a unique bridge, then for every $j \in \{1, 2, \dots, i-1\}$, the path π must use at least one edge from the j -th copy of the unique bridge; otherwise it would be impossible to reach the i -th copy of w . Among all edges used by π , consider the last one that is from the second copy of the unique bridge in w , and let σ be the suffix of π that begins in this edge. This suffix σ cannot use any vertices from the first copy of w , and therefore we can shift σ by one copy to the left, and get a legitimate inner path in w^n , call it σ_{\leftarrow} . The first edge used by σ_{\leftarrow} is from the first copy of the unique bridge. The path π must use some edge from the first copy of the unique bridge, and this edge can be connected by an inner path to the first edge used by σ_{\leftarrow} , since they both belong to the first copy of the unique bridge. Summing up, we have shown that the target vertex of π can be shifted by one copy to the left, without affecting reachability via an inner path from the s -th left port, thus proving (3).

By the same kind of monotonicity argument as in the proof of Claim 3.18, one can show that there is some threshold such that

$$S_i^n = S_i^{n+1} \quad (4)$$

holds for all $i < n$ such that the distance from n to i exceeds the threshold. For $i \in \{1, 2, \dots\}$, let us write S_i for the value of S_i^n for some (equivalently, every) n whose distance to i exceeds the threshold. By (3), we know that the sequence S_1, S_2, \dots is decreasing, and therefore it must eventually reach some stable value, which we denote by S . Define T_i^n in the same way as S_i^n , except that the t -th right port is used instead of the s -th left port. By the same argument as for S_i^n , there is some stable value T such that $T = T_i^n$ holds for all $i < n$ such that the distance from 1 to i exceeds the threshold. Consider now a number n that is bigger than the sum of both thresholds (the threshold for S and the threshold for T). If we take i to be half of n , rounded up, then

$$S = S_i^n \quad \text{and} \quad T = T_i^n.$$

If the sets S and T have nonempty intersection, then w^n admits an inner path from the s -th left port to the t -th right port; if the sets are disjoint then there is no such path. In other words, the fixed sets S and T determine the answer to the question in the claim for all large enough n , thus proving the claim. \square

The purpose of the above lemma is to get the following corollary.

Corollary 3.20. *If w is a context with at most one bridge, then $\alpha(w)$ is aperiodic for every homomorphism $\alpha : P_k \rightarrow A$ that is aperiodic modulo reachability.*

Proof:

This is an almost immediate application of Lemma 3.17 and the definition of aperiodicity modulo reachability, with one caveat. Let β be the reachability homomorphism. Apply Lemma 3.17 to the context w , yielding aperiodicity of $\beta(w)$. This almost matches the assumption in the implication from the definition of aperiodicity modulo reachability, which says that

$$\beta(w) \text{ is idempotent} \quad \Rightarrow \quad \alpha(w) \text{ is aperiodic.}$$

The difference is that we know that $\beta(w)$ is aperiodic, while the implication requires the stronger property of being idempotent. However, the weaker assumption is enough, as shown in the following claim.

Claim 3.21. If α is aperiodic modulo reachability, then every context $w \in P_k$ satisfies

$$\beta(w) \text{ is aperiodic} \quad \Rightarrow \quad \alpha(w) \text{ is aperiodic}$$

Proof:

Suppose that $\beta(w)$ is aperiodic. This means that for sufficiently high powers n , $\beta(w^n)$ is an idempotent, and therefore $\alpha(w^n)$ is aperiodic. In other words, we have established that all sufficiently high powers of $\alpha(w)$ are aperiodic. To conclude that $\alpha(w)$ is aperiodic, we use the following observation about finite monoids:

- (*) If a is an element of a finite monoid such that some two consecutive powers a^n and a^{n+1} are aperiodic, then also a is aperiodic.

To see why (*) is true, we observe that all sufficiently large m satisfy

$$a^{m+n+1} \quad \underbrace{=} \quad a^m \quad \underbrace{=} \quad a^{m+n},$$

because a^{n+1} is aperiodic because a^n is aperiodic

which implies that all sufficiently large m satisfy $a^m = a^{m+1}$, thus establishing aperiodicity of a . \square

3.3.2. The induction

Having established aperiodicity of contexts with at most one bridge, we return to the proving the harder implication of Theorem 3.9, which says that if a language is recognized by a homomorphism that is aperiodic modulo reachability, then it is star-free. Consider a homomorphism

$$\alpha : P_k \rightarrow A,$$

which is aperiodic modulo reachability. We want to show that every language recognized by α is star-free. Without loss of generality, we can assume that α refines the reachability homomorphism. Indeed, if we add the outputs of the reachability homomorphism to a homomorphism α , then the new

homomorphism will refine reachability, will still be aperiodic modulo reachability, and will recognize all languages recognized by the original homomorphism α .

Fix a homomorphism α for the rest of this section. We assume that it is aperiodic modulo reachability, it refines the reachability homomorphism, and it is surjective. We will show that every language recognized by α is star-free. We say that $a \in A$ is star-free if the inverse image $\alpha^{-1}(a)$ is star-free. We will show that every element of A is star-free; this will immediately imply that every language recognized by α is star-free, since every such language is a finite union of inverse images $\alpha^{-1}(a)$.

Green's relations. In the proof, we will use Green's relations, so we begin with a quick summary of these relations and their basic properties. Green's relations are three pre-orders on elements $a, b \in A$, which are defined as follows (the terminology of infixes, prefixes and infixes taken from [6, Section 1.2]):

$$\underbrace{a \in b \cdot A}_{b \text{ is a prefix of } a} \quad \underbrace{a \in A \cdot b}_{b \text{ is a suffix of } a} \quad \underbrace{a \in A \cdot b \cdot A}_{b \text{ is an infix of } a}.$$

All of these relations are pre-orders, i.e. they are transitive and reflexive, but not necessarily anti-symmetric. Two monoid elements are called *prefix equivalent* if they are prefixes of each other (in other words, they generate the same right ideals $a \cdot A = b \cdot A$), likewise we define suffix equivalence and infix equivalence. Define an \mathcal{H} -class to be a nonempty intersection of some prefix class with some suffix class. We will be particularly interested in \mathcal{H} -classes which contain an idempotent, i.e. a monoid element e that satisfies $e = e \cdot e$. The following lemma, see [6, Lemmas 1.11–1.14], sums up the properties of Green's relations that will be used in our proofs.

Lemma 3.22. (Green's Relations Lemma)

The following properties hold in every finite monoid:

1. If a is a prefix of b , and a is infix equivalent to b , then b is a prefix of a ; likewise for suffixes.
2. All \mathcal{H} -classes contained in the same infix class have the same size.
3. If an \mathcal{H} -class contains an idempotent, then it is a group.

The induction. Having described Green's relations, we resume the proof that every element of A is star-free. The proof is by induction on the position of the element in the infix ordering. Consider some infix class $J \subseteq A$, and assume that we have already shown star-freeness for every monoid element that is a strict infix of some (equivalently, every) element of J . We will show that also all monoid elements in J are star-free. The proof will consider three possible cases, which are identified in the following lemma.

Lemma 3.23. Let $\alpha : P_k \rightarrow A$ be a surjective monoid homomorphism, which refines the reachability homomorphism, and which is aperiodic modulo reachability. Every infix class $J \subseteq A$ satisfies one of the following conditions:

1. J contains no idempotent; or

2. every context in $\alpha^{-1}(J)$ has at least two bridges; or
3. J is \mathcal{H} -trivial, i.e. every \mathcal{H} -class contained in J has size one.

Proof:

To prove the lemma, we will show that if an infix class J contains an idempotent and there is some context in $\alpha^{-1}(J)$ that has at most one bridge, then J is \mathcal{H} -trivial. Let J be such an infix class. We begin with the following claim, which shows that the number of persistent ports is an invariant of an infix class.

Claim 3.24. All contexts in $\alpha^{-1}(J)$ have the same number of persistent ports.

Proof:

Since α refines the reachability homomorphism, and the reachability homomorphism stores information about which ports are persistent, it follows that the image under α tells us which ports are persistent. If we compose two contexts, then the context can only have fewer persistent ports. It therefore follows that the invariant “number of persistent ports” can only decrease when composing contexts; and therefore this invariant must be constant within a single infix class. \square

The following claim will show that having at most one bridge is also an invariant of the infix class.

Claim 3.25. Let v and w be contexts with the same number of persistent ports, such that v is an infix of w . If v has at most one bridge, then so does w .

Proof:

Let f be the function which maps vertices of v to their corresponding vertices in w , when v is viewed as an infix of w . This function maps non-port vertices to non-port vertices, and therefore it maps inner paths of v to inner paths of w . Since v and w have the same number of persistent ports, the function f does not change persistence:

$$x \text{ is a persistent vertex in } v \quad \Leftrightarrow \quad f(x) \text{ is a persistent vertex in } w.$$

Using these two observations, we prove the claim. Consider an inner path π in w which connects some non-persistent left port with some non-persistent right port. This path must necessarily contain a segment which is the image, under f , of some path σ in v that goes from a left port of v to a right port of w . Since f does not change persistence, it follows that σ must use a vertex from the unique bridge in v . Therefore, π must use a vertex from the image of this unique bridge under f , and this image is contained in a unique component of w . \square

Putting the above two claims together with the assumption that some element of J is the image of a context with at most one bridge, we conclude that every element of J is the image of a context with at most one bridge. By Corollary 3.20 we know that all elements of J are aperiodic. In particular, every group contained in J has size one, since a group with only aperiodic elements is necessarily of size one. The infix class contains at least one \mathcal{H} -class which is a group, since it contains an idempotent (see item 3 of Lemma 3.22). Since this \mathcal{H} -class has size one, all other \mathcal{H} -classes also have size one, because all \mathcal{H} -classes in a given infix class have the same size, see item 2 of Lemma 3.22. \square

Using the above lemma, it is enough to show the induction step for an infix class which satisfies one of the three conditions in the lemma. These cases are treated in Sections 3.3.3–3.3.5 below. The case of at least two bridges is unique to graphs, while the remaining two cases (no idempotents or trivial \mathcal{H} -classes) are treated the same way as in Schützenberger’s proof that aperiodic monoids recognize only star-free word languages.

3.3.3. No idempotents

Assume that $J \subseteq A$ is an infix class without any idempotent. In monoid theory, such an infix class is called *non-regular*, and such infix classes are known to be simple because they necessarily decompose into strictly smaller infix classes, as explained in the following lemma.

Lemma 3.26. If J is an infix class without idempotents, then every context in $\alpha^{-1}(J)$ can be decomposed as $w_1 w_2 w_3$ such that w_2 is a k -generator and both w_1 and w_3 have images under α that are strict infixes of J .

Proof:

Consider a context w with image in J , and view it as a word whose letters are k -generators. Let w_1 be the longest prefix of this word with image not in J , let w_2 be the letter just after w_1 , and let w_3 be the rest of the word. If w_3 would have image in J , then $J \cdot J$ would contain an element of J , which cannot happen in an infix class without an idempotent, see [15, Corollary 2.25]. \square

The above lemma implies the sufficient condition in the following lemma, with $\ell = 3$, and thus proves star-freeness of every element in J .

Lemma 3.27. The following condition is sufficient for star-freeness of $a \in A$: every monoid element that is a strict infix of a is star-free, and there is some $\ell \in \{1, 2, \dots\}$ such that every context in $\alpha^{-1}(a)$ can be decomposed as $w_1 \cdots w_\ell$ such that each w_i is either a k -generator, or $\alpha(w_i)$ is a strict infix of a .

Proof:

The contexts in $\alpha^{-1}(a)$ are described by the star-free expression

$$\bigcup_{\substack{a_1, \dots, a_\ell \\ a = a_1 \cdots a_\ell}} L_{a_1} \cdots L_{a_\ell}$$

where L_b is defined to be the following star-free language:

$$L_b = \begin{cases} \alpha^{-1}(b) & \text{when } b \text{ is a strict infix of } a; \\ k\text{-generators in } \alpha^{-1}(b) & \text{otherwise.} \end{cases}$$

\square

3.3.4. Trivial \mathcal{H} -classes

Consider now an infix class J where all \mathcal{H} -classes are trivial. This part of the proof is the same as in Schützenberger original proof, where a monoid was considered that had only trivial \mathcal{H} -classes. We begin with the following lemma, which says that if we are given a context whose image is known to be in J , then we can use star-free expressions to determine that value.

Lemma 3.28. Every $a \in J$ is star-free over $\alpha^{-1}(J)$ in the following sense: there is a star-free expression that coincides with $\alpha^{-1}(a)$ over contexts from $\alpha^{-1}(J)$.

Proof:

Assume that w has image in J . Under this assumption, we can use item 1 of Lemma 3.22 to infer that $\alpha(w)$ is prefix equivalent to $a \in J$ if and only if it has a prefix with this property. By looking at a minimal prefix with this property, we see that $\alpha(w)$ is prefix equivalent to a if and only if w belongs to the star-free language

$$\bigcup_{b,c} \alpha^{-1}(b) \cdot (k\text{-generators with value } c) \cdot P_k,$$

where the sum ranges over choices of $b, c \in A$ such that b is a strict infix of J and bc is prefix equivalent to a . In other words, there is a star-free language which coincides with the (inverse image of the) prefix class of a over contexts from $\alpha^{-1}(J)$. Using a symmetric result for suffix classes, we see that there is a star-free language that coincides with the (inverse image of the) \mathcal{H} -class of a over contexts with image in J . Since this \mathcal{H} -class is assumed to be trivial, we get the conclusion of the lemma. \square

The above lemma, together with the following one, implies that all elements of J are star-free.

Lemma 3.29. The language $\alpha^{-1}(J)$ is star-free.

Proof:

Let L be the contexts whose image under α is not an infix of J . In other words, these are the contexts whose image under α is – with respect to the infix ordering – either strictly bigger than J or incomparable to it. If we take the complement of L , and then we remove the contexts with images that are strict infixes of J , which are star-free by induction assumption, then we are left with the language from the statement of the lemma. Therefore, it is enough to show that L is star-free. Using the same argument as in Lemma 3.28, we see that L is defined by the expression

$$\bigcup_{b,c} \alpha^{-1}(b) \cdot (k\text{-generators with value } c) \cdot P_k, \tag{5}$$

where the sum ranges over choices of $b, c \in A$ such that b is an infix (not necessarily strict) of J and bc is not an infix of J . The above expression is not yet known to be star-free, since it uses subexpressions of the form $\alpha^{-1}(b)$ for $b \in J$. However, such a subexpression can be replaced by star-free languages without affecting the value of the entire expression, as follows.

By Lemma 3.28 and the induction assumption, for every $b \in J$ there is a star-free language L_b which coincides with $\alpha^{-1}(b)$ on contexts whose image is an infix of J . In other words, L_b is equal to $\alpha^{-1}(b)$ plus some extra contexts which are in L . In the expression (5), replace each sub-expression $\alpha^{-1}(b)$ with $b \in J$ by the expression L_a . The extra contexts from the new sub-expressions are outside L , and therefore after this replacement the expression will still define L . \square

3.3.5. At least two bridges

We are left with the case when every context with image in J has at least two bridges. This case will be resolved using the following lemma.

Lemma 3.30. If a context in P_k has at least two bridges, then it can be decomposed as $w_1 \cdots w_\ell$ so that $\ell \leq \mathcal{O}(k)$ and each of the contexts w_1, \dots, w_ℓ is either a k -generator, or has strictly more persistent ports than w .

Before proving the lemma, we use it to complete the proof of Theorem 3.9, by showing that every element in J is star-free, assuming that every context with image in J has at least two bridges. As we have observed at the beginning of the proof of Lemma 3.23, since the homomorphism α refines the reachability homomorphism, it follows that the number of persistent ports is an invariant of the infix class J . Therefore, in the decomposition from Lemma 3.30, each factor w_i is either a k -generator, or its image is a strict infix of J . Hence, we can apply Lemma 3.27 to conclude that every element of J is star-free.

To complete the proof of Theorem 3.9, it remains to prove Lemma 3.30. One of the main ingredients of this proof will be a result based on [2] and [8], see Lemma 3.31 below, which says that a path decomposition can be modified without affecting its width so that it does not alternate too much between different inner components. To state this lemma, we need to introduce some terminology for path decompositions. Recall that a path decomposition is a sequence of sets of vertices, called bags. This sequence might contain repetitions, i.e. the same set of vertices might appear several times in the sequence. To avoid ambiguity use the word bag to describe an index in the sequence, and not the set of vertices that is found at this index; and hence different bags might contain the same vertices. Recall that an interval in a path decomposition is a set of bags which forms an interval with respect to the total ordering on bags. We say that a vertex is *active* in an interval if it is added or removed (or both) in the interval, i.e. some bags from the interval have this vertex and some do not.

Lemma 3.31. (Dealternation Lemma)

Let w be a context and let $X \cup Y$ be a partition of its non-port vertices such that there is no edge between X and Y . If w has a path decomposition of width k , then it also has a path decomposition of width k which can be partitioned into $\mathcal{O}(k)$ intervals such that in each interval at most one of the sets X or Y is active.

Proof:

This lemma is essentially proved in [8, Lemma 17], which itself is based on the typical sequences from [2, p. 365]. The only purpose of this proof is to introduce sufficient terminology so that we can apply the cited results. Fix some context w of pathwidth k for the rest of this proof; all path

decompositions in the proof will be path decompositions of this context. A path decomposition of w can be viewed as a sequence of instructions from the set

$$\{\text{add}(x), \text{remove}(x) : x \text{ is a vertex of } w\}$$

such that: (a) every vertex is added at most once and removed at most once in the sequence, and it cannot be added after having been removed; (b) a vertex is added if and only if it is not a left port; (c) a vertex is removed if and only if it is not a right port. In particular, persistent ports are neither added nor removed. An instruction sequence describes a path decomposition in the following way. The bags of the path decomposition are prefixes of the instruction sequence, including the empty and full prefixes. The contents of such a bag is the set of vertices that is obtained by starting with the left ports, and then executing all instructions in the prefix corresponding to the bag. Condition (b) ensures that the first bag contains exactly the left ports; likewise for condition (c) and the last bag containing the right ports. All path decompositions arise this way, assuming that we consider path decompositions where consecutive bags differ by at most one vertex. To be a valid path decomposition, the instruction sequence needs to furthermore satisfy (d) for every edge there is some prefix whose corresponding bag contains both endpoints of the edge.

To prove the lemma, we will begin with some path decomposition of width k , viewed as an instruction sequence in the sense described above, and then we will modify it to achieve the conclusion of the lemma using a permutation that respects the order within each of the sets X and Y . Define a permutation of instructions in an instruction sequence to be *separated* if the only instruction pairs whose mutual order is changed by the permutation are pairs where one instruction operates on X and the other instruction operates on Y . It is easy to see that if an instruction sequence is a path decomposition, then applying a separated permutation also yields a path decomposition; here it is important that there are no edges connecting X and Y . We are now ready to apply [8, Lemma 17], which says that for every instruction sequence, one can apply a separated permutation so that in the new instruction sequence: (i) the width of the corresponding path decomposition does not increase; and (ii) the instructions can be grouped into $\mathcal{O}(k)$ intervals which operate either only on X , or only on Y , or only on ports. The resulting instruction sequence corresponds to a path decomposition as required in the current lemma. \square

Equipped with the Dealternation Lemma, we can prove Lemma 3.30.

Proof:

It is not hard to see that the lemma is equivalent to the following statement: (*) every context $w \in \mathcal{P}_k$ with at least two bridges has a path decomposition of optimal width, whose bags can be partitioned into $\mathcal{O}(k)$ intervals so that in every interval there is some vertex that is present in all bags of the interval and which is not a persistent port of w . The intervals in (*) correspond to the contexts w_1, \dots, w_ℓ , and the vertices present in all bags correspond to their new persistent ports. It remains to prove (*).

Suppose first that w has an edge that goes directly from a non-persistent left port x to a non-persistent right port y . Every path decomposition of w – including those of optimal width – must have some bag that contains both endpoints of this edge; all bags to the left of this bag contain x and all bags to the right of this bag contain y . This proves (*) and therefore also the lemma for contexts which have bridge that is a single edge.

Consider now a context that has at least two bridges, but which does not have any edge as discussed in the previous paragraph. Take one of the bridges, and let X be the non-port vertices that are incident to this bridge. Let Y be the remaining non-port vertices. Since there are no edges connecting X and Y , we can apply Lemma 3.31, yielding an optimal width path decomposition and a partition of its bags into a family of at most $\mathcal{O}(k)$ intervals, call this family \mathcal{I} . By refining this family, we can assume without loss of generality that every port is inactive in every interval from \mathcal{I} . To prove (*), we need to show that for every interval $I \in \mathcal{I}$, there is a vertex that is not persistent port of w and which appears in all bags of I . If I uses some non-persistent port vertex at least once, then this port vertex is present in all bags of I and we are done. The interesting case is when I does not use any port vertices except for the persistent ports.

We know that all non-persistent left ports of w are to the left of the interval I , and all non-persistent right ports of w are to the right of the interval. Since one of the bridges requires using a vertex from X , and one of the bridges requires using a vertex from Y , it follows that the interval must use at least one vertex from X and at least one vertex from Y . By the Dealternation Lemma, either X or Y is inactive in the interval, and the inactive set will contribute a vertex that is present in all bags of the interval, thus proving (*). \square

4. Future work

We finish the paper with some potential directions for future work.

1. **Directed graphs.** It is not clear how to generalize separator logic to directed graphs.
2. **Cliquewidth.** The star-free expressions used in this paper are based on operations designed for treewidth. There are also operations designed for cliquewidth, and it is natural to ask about a variant of first-order logic that is equivalent to star-free expressions for these operations.
3. **Bounded treewidth.** It would be nice to generalize Theorem 3.9 so as to get an algebraic characterization of star-free languages of bounded treewidth. As mentioned previously, the problem is open already for trees.
4. **Other algebraic characterizations for bounded pathwidth.** Over bounded pathwidth, one could attempt algebraic characterizations of other logics. One natural candidate is the usual variant of first-order logic with the edge relation only; in fact an algebraic characterization of this logic could even be attempted for bounded treewidth, as there corresponding logic for trees is already understood [1, Theorem 1]. Another natural candidate is the extension of first-order logic with predicates for disjoint paths from [16, Section 4].

References

- [1] Michael Benedikt and Luc Segoufin. Regular tree languages definable in FO and in FO_{mod} . *ACM Trans. Comput. Log.*, 11(1):4:1–4:32, 2009.

- [2] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.
- [3] Mikołaj Bojańczyk. Forest Expressions. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 146–160. Springer, 2007.
- [4] Mikołaj Bojańczyk. Some open problems in automata and logic. *ACM SIGLOG News*, 2(4):3–15, 2014.
- [5] Mikołaj Bojańczyk. Two monads for graphs. *CoRR*, abs/1804.09408, 2018.
- [6] Mikołaj Bojańczyk. Languages recognised by finite semigroups, and their generalisations to objects such as trees and graphs, with an emphasis on definability in monadic second-order logic. <https://www.mimuw.edu.pl/~bojan/papers/algebra-26-aug-2020.pdf>, 2020.
- [7] Mikołaj Bojańczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 407–416. ACM, 2016.
- [8] Mikołaj Bojańczyk and Michal Pilipczuk. Optimizing Tree Decompositions in MSO. In Heribert Vollmer and Brigitte Vallée, editors, *Symposium on Theoretical Aspects of Computer Science, STACS, Hannover, Germany*, volume 66 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
- [9] John A. Brzozowski and Imre Simon. Characterizations of locally testable events. *Discrete Math.*, 4:243–271, 1973.
- [10] Bruno Courcelle. A representation of graphs by algebraic expressions and its use for graph rewriting systems. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science*, pages 112–132, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.
- [11] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3), June 2017.
- [12] Thilo Hafer and Wolfgang Thomas. Computation tree logic ctl^* and path quantifiers in the monadic theory of the binary tree. In *International Colloquium on Automata, Languages and Programming, ICALP, Turku, Finland*, pages 269–279. Springer, 1987.
- [13] J.A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California, Los Angeles, 1968.
- [14] Robert McNaughton and Seymour Papert. *Counter-free automata*. The M.I.T. Press, Cambridge, Mass.-London, 1971.
- [15] Jean-Eric Pin. Mathematical foundations of automata theory.
- [16] Nicole Schrader, Sebastian Siebertz, and Alexandre Vigny. First-order logic with connectivity operators. <https://arxiv.org/abs/2107.05928>, 2021.
- [17] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.
- [18] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser. 1994.