Graph Theory in Coq: Minors, Treewidth, and Isomorphisms

Christian Doczkal, Damien Pous

Graph Theory in Coq: Minors, Treewidth, and Isomorphisms*

Christian Doczkal · Damien Pous

October 15, 2019

Abstract We present a library for graph theory in Coq/Ssreflect. This library covers various notions on simple graphs, directed graphs, and multigraphs. We use it to formalise several results from the literature: Menger's theorem, the excluded-minor characterization of treewidth-two graphs, and a correspondence between multigraphs of treewidth at most two and terms of certain algebras.

Keywords graph theory, minor, treewidth, isomorphisms, Coq, Ssreflect

1 Introduction

Despite the importance of graph theory in mathematics and computer science, there are only a few formalizations of graph theory results in interactive theorem provers, and even fewer general purpose libraries.

In the 1990s, Chou formalized some basic undirected graph theory (paths, connectedness, trees) in HOL [2] and used it for the verification of distributed algorithms [3]. Nakamura and Rudnicki formalized Euler's theorem in Mizar [24]. In the 2000s, several authors formalized results about planar graphs: in addition to Gonthier's celebrated formal proof of the Four-Color Theorem [17], planar graphs were formalized in Isabelle/HOL for the Flyspeck project [25] and Durfourd and Bertot employed a notion of graphs based on hypermaps embedded in a plane to study Delaunay triangulations [14]. More recently, Noschinski developed a library for both simple and multi-graphs in Isabelle/HOL [26].

* This paper extends and revises the results presented in [11]; the underlying Coq library is available from `https://perso.ens-lyon.fr/damien.pous/covece/graphs/`.

This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

C. Doczkal, D. Pous
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France

All in all, a general purpose graph library in currently missing in Coq, and we propose one in the present paper. The library deals with simple graphs, directed graphs, and multigraphs; it currently includes basic notions like paths, trees, subgraphs, separators, and isomorphisms, as well as a few more advanced ones: minors, and treewidth, whose theory was never formalized to the best of our knowledge. We use the library to formalize three distinct results, which we discuss below.

Menger's theorem. Menger's Theorem [23] states that if one needs to remove at least n vertices to disconnect two sets of vertices A and B of some graph, then there exist n pairwise disjoint paths from A to B. Diestel [8, p. 50] calls Menger's Theorem [23] one of the cornerstones of graph theory and remarks that Hall's Marriage Theorem [20], a straightforward consequence of Menger's Theorem, is one of the most applied graph-theoretic results [8, p. 42].

Menger's Theorem provides a good test-case for our graph library: it admits a very short paper proof [18], but it nevertheless requires tools to work efficiently with several basic concepts like paths (including collections of paths), and deleting vertices and edges from graphs. We prove the theorem for directed graphs and use it to derive several corollaries on simple graphs as well as multigraphs: this ensures that its formulation is general enough and that our infrastructure makes it possible to transfer results between different kinds of graphs with minimal effort.

Excluded-minor characterization for treewidth two. The notion of *treewidth* [8] measures how close a graph is to a forest. Graph homomorphism (and thus k-coloring) becomes polynomial-time for classes of graphs of bounded treewidth [15, 1, 19], so does model-checking of Monadic Second Order (MSO) formulae, and satisfiability of MSO formulae becomes decidable, even linear [6, 7].

Robertson and Seymour's graph minor theorem [29], a cornerstone of algorithmic graph theory, states that graphs are well-quasi-ordered by the *minor* relation. As a consequence, the classes of graphs of bounded treewidth, which are closed under taking minors, can each be characterized by finitely many excluded minors. While the graph minor theorem is nonconstructive and does not yield the excluded minors, low-width instances where known before: the graphs of treewidth at most one (the forests) are precisely those excluding the cycle with three vertices ($\mathsf{C_3}$); those of treewidth at most two are those excluding the complete graph with four vertices ($\mathsf{K_4}$) [13].


($\mathsf{C_3}$)

($\mathsf{K_4}$)

We build on our library to present a constructive and formal proof of the latter result in Coq/Ssreflect. Unlike in conference version of this paper [11], we present here a direct proof relying on Menger's Theorem.

Multigraphs of treewidth two seen as an algebra. Amongst the open problems related to treewidth, there is the question of finding finite axiomatisations of isomorphism for graphs of a given treewidth [7, page 118]. This question was recently answered positively for treewidth two [22, 12]:

Two-pointed $\mathsf{K_4}$-free multigraphs form the free $2p$-algebra, (†)

where 2p-algebras are algebraic structures characterized by twelve equational axioms. The proof is rather technical; it builds on a precise analysis of the structure of K_4-free graphs. Further, invalid proofs of related claims have already been published in the literature (see [22]).

Our initial motivation for this work [11] was to formalize (†): not only will this give us assurance about the validity of the proof in [22], it will also allow for the development of automation tactics for certain algebraic theories (e.g., 2p-algebra, allegories [16,28]).

We present three important steps towards the formalisation of this result:

1. graphs form a 2p-algebra;
2. graphs of treewidth at most two form a subalgebra;
3. every graph excluding K_4 as a minor can be represented by a term.

These three steps cover different situations which are frequently encountered when reasoning about graphs. The first point requires us to define several constructions on graphs (e.g., disjoint unions and quotients), and to establish a number of isomorphisms. The second point requires us to show that these constructions preserve treewidth. The third point requires us to analyse the structure of K_4-free graphs in order to decompose them recursively; for this we again make use of Menger's theorem.

The final step for (†) consists in proving that terms denoting isomorphic graphs are equivalent modulo the axioms of 2p-algebras. We leave the formalisation of this step for future work.

Outline. We discuss our representation choices for directed graphs, simple graphs, and paths in Section 2. Then we proceed to proving Menger's Theorem (Section 3). We define minors, tree decompositions and treewidth in Section 4, so that we can prove the minor-exclusion theorem for treewidth two in Section 5. We end the presentation of the general purpose part of the library by defining labeled multigraphs in Section 6, together with the associated notion of isomorphism.

In the remaining sections we apply our library to 2p-algebra: We define a 2p-algebra of graphs and show that treewidth at most two graphs form a subalgebra (Section 7). We then prove specific properties of K_4-free graphs (Section 8), which make it possible to recursively extract terms from connected K_4-free graphs (Section 9). We finally extend this extraction function to arbitrary K_4-free graphs in Section 10.

Differences with [11]. The Coq library accompanying this paper [10] evolved significantly since [11] and continues to evolve, whence the different structure of the present paper: the main contribution is the graph library itself, rather than the application to 2p-algebras.

The most noticeable improvement is that we prove Menger's Theorem and use it both to provide a direct proof of the minor-exclusion theorem for treewidth two, and to simplify the recursive analysis of K_4-free graphs. (Lemma 8.9 follows easily with Menger's Theorem, while it was requiring a long series of ad-hoc lemmas in [11].) We also prove that graphs satisfy the laws of 2p-algebra (Lemmas 7.2 and 7.3). This required us develop techniques for dealing with the complicated isomorphisms arising with nested quotients and disjoint unions in a compositional manner (Lemma 6.6).

2 Graphs and Paths

In this section we describe how we represent finite graphs in Coq. The representation is based on finite types as defined in the mathematical components library [31]. We start by briefly introducing finite types and the notations we are going to use in the mathematical development.

If X and Y are types, we write $X + Y$ for the *sum type* (with elements $\mathsf{inl}\, x$ and $\mathsf{inr}\, y$), $X \times Y$ for the *product type* (with elements (x, y)), and $X_\bot$ for the *option type* (with elements $\mathsf{Some}\, x$ and None). For indexed type families $T : X \to \mathsf{Type}$, we write $\Sigma(x : X).\, T\, x$ for the *sigma type* (i.e., the type of dependent pairs) with elements $\langle x, y\rangle$ where $x : T$ and $y : T\, x$.

As usual, we write $g \circ f$ for the composition of two functions f and g. For functions f and g, we write $f \equiv g$ to mean that f and g agree on all arguments.

A *finite type* is a type X together with a list enumerating its elements. Examples of finite types are the type $\mathbb{B}$ of booleans, and the type I_n of natural numbers smaller than n. Finite types are closed under many type constructors. In particular, they are closed under sums, products, and sigma types. If X is a finite type, we write 2^X for the finite type of sets over X with decidable membership. For sets $A : 2^X$, we write $\overline{A}$ for complement of A in X. We slightly abuse notation and also write X for the full set over some type X. In particular, we use $|\_|$ to denote the cardinality of both sets and finite types (e.g., $|I_n| = n$). Finite sets come with an operation $\mathsf{pick} : 2^X \to X_\bot$ where $\mathsf{pick}\, A = \mathsf{Some}\, x$ for some $x \in A$ if A is nonempty and $\mathsf{pick}\, A = \mathsf{None}$ otherwise. If $\approx : X \to X \to \mathbb{B}$ is a boolean equivalence relation, the *quotient* [4] of X with respect to $\approx$, written $X_{/\approx}$, is a finite type as well. The type $X_{/\approx}$ comes with functions $\pi : X \to X_{/\approx}$ and $\overline{\pi} : X_{/\approx} \to X$ such that $\pi(\overline{\pi}\, x) = x$ for all $x : X_{/\approx}$ and $\overline{\pi}(\pi\, x) \approx x$ for all $x : X$.

We use finite types as the basic building block for defining (finite) graphs.[1]

Definition 2.1 A *(finite) directed graph*, or *digraph* for short, is a structure $\langle V, R\rangle$ where V is a finite type of vertices and $R : V \to V \to \mathbb{B}$ is a decidable (i.e., boolean) *edge relation*. A *simple graph* is a digraph whose edge relation is symmetric and irreflexive.

In Coq, we represent finite digraphs and simple graphs using dependently typed records:

```
Record digraph := { vertex   :> finType;
                    edge     :  rel vertex }.

Record sgraph := { svertex  :  finType;
                   sedge    :  rel svertex;
                   sg_sym   :  symmetric sedge;
                   sg_irrefl :  irreflexive sedge }.
```

Note that for simple graphs two of the components are propositions. We introduce a coercion from simple graphs to digraphs forgetting symmetry and reflexivity of the edge relation. This allows us to define notations and notions like paths on digraphs, the class of graphs with the least structure, and have simple graphs inherit these notations and notions through the coercion. Declaring the vertex

[1] Most of the "Definition" and "Lemma" and "Theorem" headers in this paper are links to the corresponding entity in the Coq development.

field of digraphs as a coercion, allows us to write $x : G$ to denote that x is a vertex of G and $|G|$ to denote the number of vertices of G (irrespective of whether G is a digraph or a simple graph). For vertices $x, y : G$ we write $x{-}y$ if there is an edge between x and y. We write $G + xy$ for the graph G with an additional xy-edge and $G - xy$ for G with any potential xy-edge removed.

For a set $U : 2^G$ of vertices of G, the *subgraph induced by* U, written $G|_U$, has as vertices the vertices in U and as edge relation the edge relation of G restricted to U. This is formalized by taking $\Sigma x : G.\, x \in U$ as the type of vertices[2] and lifting the edge relation accordingly. While, technically, the vertices of G and $G|_U$ have different types, we will ignore this in the mathematical presentation. In Coq, we have a generic projection from $G|_U$ to G. For the converse direction we, of course, need to construct dependent pairs of vertices $x : G$ and proofs of $x \in U$.

Remark 2.2 The constructions $G|_U$ and $G + xy$ (for a given graph G) behave very differently in proofs. As mentioned above, the type of vertices of $G|_U$ is different from the type of vertices of G. This is not the case for $G + xy$ (or any construction changing only the edge relation). As a consequence, $x : G$ iff $x : G + xy$ and likewise for sets or lists of vertices. This makes an explicit conversion of the vertices unnecessary.

Almost every argument in graph theory involves paths in one form or another. Consequently, finding the right representation of paths is of utmost importance when formalizing graph theory.

Definition 2.3 Let G be a digraph. An *xy-path* is a nonempty sequence of vertices p beginning with x and ending with y such that $z{-}z'$ for all adjacent elements z and z' of p (if any). A path is *irredundant* if all vertices on the path are distinct.

The Mathematical Components library includes a predicate and a function

$$\mathsf{path} : \forall\ \mathsf{T} : \mathsf{Type}.\ \mathsf{rel}\ \mathsf{T} \to \mathsf{T} \to \mathsf{seq}\ \mathsf{T} \to \mathbb{B} \qquad \mathsf{last} : \forall\ \mathsf{T}.\ \mathsf{T} \to \mathsf{seq}\ \mathsf{T} \to \mathsf{T}$$

such that $\mathsf{path}\, e\, x\, q$ holds if the list $x :: q$ represents a path in the relation e, and $\mathsf{last}\, x\, q$ returns the last element of $x :: q$. The functions path and last account for the nonemptiness of paths though the use of two arguments: the first vertex x and the (possibly empty) list of remaining vertices q.

Thus, the notion of an xy-path (in some fixed digraph G) can be formalized as a predicate on lists:

$$\mathsf{pathp}\, x\, y\, p := \mathsf{path}\, (\mathsf{edge}\, G)\, x\, p \wedge \mathsf{last}\, x\, p = y$$

However, this is cumbersome to use for several reasons: First it leads to many different assumptions, i.e., both $p : \mathsf{seq}\, G$ and $\mathsf{pathp}\, x\, y\, p$ for a single xy-path. This is inconvenient since a single lemma can easily involve six or more different paths, thus cluttering the context. Second, "u is a vertex of the xy-path p" would be written as $u \in x :: p$, i.e., requiring explicit mention of the first vertex. Lastly and most importantly, the asymmetric definition gets in the way of symmetry reasoning by path reversal (in simple graphs). For instance, consider some p satisfying

[2] To be fully precise, we use the type $\Sigma x : G.\, x \in U = \mathsf{true}$, exploiting that set membership is decidable. Since equality between booleans is proof irrelevant (i.e., there is at most one proof of $x \in U = \mathsf{true}$), this ensures that $|G|_U| = |\Sigma x : G.\, x \in U| = |U|$. This is a standard technique used pervasively in the mathematical components library.

$\mathsf{pathp}\,x\,y\,p$. Then the corresponding p' satisfying $\mathsf{pathp}\,y\,x\,p'$ is obtained by removing the last element of $x :: p$ and reversing the result. In particular p' cannot be computed from p alone.

We solve these problems by packaging the predicate pathp into an indexed family of types:

$$\textbf{Record}\ \mathsf{Path}\ \mathsf{x}\ \mathsf{y} := \{\ \mathsf{pval} : \mathsf{seq}\ \mathsf{G}\ ;\ \mathsf{pvalP} : \mathsf{pathp}\ \mathsf{x}\ \mathsf{y}\ \mathsf{pval}\ \}$$

This allows us to endow paths with their own membership operation taking care of the fact that the list representing a path does not contain the first vertex. Moreover, it abstracts from the asymmetry in the definition of the path predicate, easing symmetry reasoning. In the following we write $x \rightsquigarrow y$ for the type $\mathsf{Path}\,x\,y$.

Indexing paths by their endpoints also allows us to define a dependently typed concatenation function "$+\!\!+$" for paths, i.e., for $\pi_1 : x \rightsquigarrow y$ and $\pi_2 : y \rightsquigarrow z$ we have $\pi_1 +\!\!+ \pi_2 : x \rightsquigarrow z$ as one would expect. We prove various trivialities about paths (e.g., $x \in \pi$ whenever $\pi : x \rightsquigarrow y$) that are then added to the hint databases of standard automation tactics. We also prove a number of lemmas for splitting paths or transferring paths between graphs. For instance, the lemma below, which allows splitting paths at the first vertex satisfying some criterion, is used more than twenty times throughout the development.

Lemma 2.4 *Let G be a digraph, $x, y : G$, $A : 2^G$, and $\pi : x \rightsquigarrow y$ such that $\pi \cap A \neq \emptyset$. Then there exist $z : G$ and $\pi_1 : x \rightsquigarrow z$ and $\pi_2 : z \rightsquigarrow y$ such that $\pi = \pi_1 +\!\!+ \pi_2$ and $\pi_1 \cap A = \{z\}$.*

Note that a path is always a path in some specific graph, i.e., if $x, y : G$ then there is an implicit G in the type $x \rightsquigarrow y$. On the other hand, a sequence of vertices can give rise to paths in different graphs, e.g., if $\pi : x \rightsquigarrow y$ is a path in G, then the underlying sequence of vertices uniquely determines a path in $G+uv$. Therefore, we introduce a function, $\mathsf{seq\_of} : \forall G (x\,y : G).\, x \rightsquigarrow y \to \mathsf{seq}\,G$ returning the underlying sequence of vertices.

Lemma 2.5 *Let G be a digraph, let $x, y, u, v : G$, and let $\pi : x \rightsquigarrow y$ be a path in G. Then there exists path $\rho : x \rightsquigarrow y$ in $G + uv$ such that $\mathsf{seq\_of}\,\rho = \mathsf{seq\_of}\,\pi$.*

The lemma above exploits that adding an edge does not change the type of vertices. In the case of induced subgraphs, where the type of vertices does change, the corresponding lemma takes a slightly different form. Writing $\lfloor \_ \rfloor$ for the natural injection from an induced subgraph to the full graph, we have:

Lemma 2.6 *Let G be a digraph, let $U : 2^G$ and let $\pi : x \rightsquigarrow y$ be a path in $G|_U$. Then there exists a path $\rho : \lfloor x \rfloor \rightsquigarrow \lfloor y \rfloor$ such that $\mathsf{seq\_of}\,\rho = \mathsf{map}\,\lfloor \_ \rfloor\,(\mathsf{seq\_of}\,\pi)$.*

The preceding lemmas can be seen as algorithmic constructions on paths. We follow the general design of the mathematical components library and prove these lemmas as propositions corresponding to the wording given (i.e., using existential quantification), allowing us to rely on existential statements from the library.

We remark that we are dealing almost exclusively with decidable properties, and for all the properties we define (e.g., connectedness of sets), we provide a boolean decider and use it when necessary (e.g., in the definition of finite sets or to justify case distinctions). Hence, if one needs to obtain a path in a computational context (i.e., when constructing inhabitants of types that are not propositions) one

can use the constructive choice operator to obtain a concrete witness; this is almost never necessary.

Paths give rise to a number of auxiliary notions. One path-based notion that is used pervasively throughout the development is that of a connected set.

Definition 2.7 Let G be a simple graph and let $A, B : 2^G$.
- A and B are *neighboring* if there exist vertices $a \in A$ and $b \in B$, such that $a{-}b$.
- A is *connected* if for every $x, y \in A$ there exists an xy-path contained in A.
- The graph G is *connected* if the full set of vertices is connected.

We prove a number of lemmas allowing us to establish connectedness of sets. In addition to a number of trivial lemmas (e.g., $\{x, y\}$ is connected whenever $x{-}y$) we also prove closure properties such as:

Lemma 2.8 *Let G be a simple graph and let $A : 2^G$ and $B : 2^G$ be connected.*
1. *If $A \cap B \neq \emptyset$, then $A \cup B$ is connected.*
2. *If A and B are neighboring, then $A \cup B$ is connected.*

The purpose of the auxiliary notion of neighboring sets is to avoid having to explicitly mention edges in certain situations and we use it repeatedly. For instance, we show once and for all that the interior of an irredundant path (i.e., the set of vertices different from the end-points) neighbors every set containing one of the end-points.

3 Menger's Theorem

Before we turn to the proof of the excluded minor characterization of treewidth-two graphs, we first prove Menger's Theorem [23]. Informally, Menger's Theorem states that if one needs to remove at least n vertices to disconnect two sets of vertices A and B of some graph, then there exist n pairwise disjoint paths from A to B. The formal statement makes use of the following definition:

Definition 3.1 Let G be a digraph and $A, B : 2^G$. A set $S : 2^G$ is an *AB-separator* if every path starting in A and ending in B contains a vertex from S. An *AB-connector* is a collection of pairwise disjoint irredundant paths such that every path starts at a vertex in A, ends at a vertex in B and has no other vertices in common with either A or B.

Menger's Theorem now formally states that the minimum size for AB-separators is also the maximum size for AB-connectors.

In Coq, there is a complication when defining the notion of connector: indexing the type of paths by the first and last vertex on the path means that in order to form a collection of paths with different endpoints we need to abstract from these endpoints. Thus we define a type of G-paths and projection functions yielding respectively the first vertex, the last vertex, and the encapsulated path:

$$\begin{aligned} G\text{-}\mathsf{path} &:= \Sigma(x, y) : G \times G.\, x \rightsquigarrow y \\ \mathsf{fst}\, \langle\langle x, y\rangle, \pi\rangle &:= x \\ \mathsf{lst}\, \langle\langle x, y\rangle, \pi\rangle &:= y \\ \mathsf{pth}\, \langle\langle x, y\rangle, \pi\rangle &:= \pi \end{aligned}$$

Note that $\mathsf{pth} : \forall \pi : G\text{-}\mathsf{path}.\, \mathsf{fst}\,\pi \rightsquigarrow \mathsf{lst}\,\pi$, i.e., the return type of $\mathsf{pth}\,\pi$ depends on the value of π. This is mainly useful in combination with predicates that are parametric in the index-vertices (e.g., $\mathsf{irred} : \forall xy : G.\, x \rightsquigarrow y \to \mathbb{B}$) or when viewing paths as sets.

With this in place, AB-connectors of size n can be defined as predicates on functions $X : I_n \to G\text{-}\mathsf{path}$. Writing X_i for the application $X\,i$, we have:

$$AB\text{-}\mathsf{connector}\, X := \forall i : I_n.\, \mathsf{irred}(X_i) \tag{1}$$
$$\wedge\ \forall i : I_n.\, X_i \cap A = \{\mathsf{fst}\, X_i\} \tag{2}$$
$$\wedge\ \forall i : I_n.\, X\,i \cap B = \{\mathsf{lst}\, X_i\} \tag{3}$$
$$\wedge\ \forall i\, j : I_n.\, i \neq j \to X_i \cap X_j = \emptyset \tag{4}$$

Before we can prove Menger's Theorem, we need two technical lemmas that respectively allow the extension of a connector with a single edge and the concatenation of two connectors.

Lemma 3.2 *Let G be a digraph, A and B sets of vertices of D, $j : I_n$ and $X : I_n \to G\text{-}\mathsf{path}$ an AB-connector. If $x \notin \bigcup_i X_i$, $\mathsf{fst}\,(X_j) = y$, $x{-}y$ and $x \notin B$, then there exists an $(\{x\} \cup Y \setminus \{y\})B$-connector of size n.*

Proof Follows by prepending x to X_j. □

Lemma 3.3 *Let G be a digraph, A and B sets of vertices of G, and P an AB-separator with $|P| = n$. Further let $X : I_n \to G\text{-}\mathsf{path}$ an AP-connector and $Y : I_n \to G\text{-}\mathsf{path}$ a PB-connector. Then there exists an AB-connector of size n.*

Proof Since all X_i (as well as all Y_i) are mutually disjoint and each contain a single vertex from P, there is for every $i : I_n$ a unique $m(i) : I_n$ such that $\mathsf{lst}\,(X_i) = \mathsf{fst}\,(Y_j)$. Since P is an AB-separator, any X_i and Y_j can intersect at most at a single vertex of P (in this case $\mathrm{j} = m(i)$). Thus, the function $Z_i := X_i +\!\!+ Y_{m(i)}$ is a connector as required.

We sketch the argument that $Z_i \cap A = \{\mathsf{fst}\,(Z_i)\}$. Assume X_i is an xy-path and $Y_{m(i)}$ is a yz-path. The inclusion from right to left is trivial, as is showing that $X_i \cap A \subseteq \{\mathsf{fst}\,(Z_i)\}$. So assume some $u \in Y_{m(i)} \cap A$. It suffices to show $u = y$ for then $u \in X_i$. This follows since the uz-part of $Y_{m(i)}$ is an AB-path and therefore must contain a vertex $v \in P$. But y is the only vertex in $P \cap Y_{m(i)}$, so $v = y = u$ since $Y_{m(i)}$ is irredundant. □

The use of "$+\!\!+$" in the definition of Z_i is a slight abuse of notation since so far we only defined concatenation for vertex-indexed paths with matching ends. In the case of G-paths, we extend the usual concatenation function with a check ensuring that the two paths do compose and only perform the concatenation in this case (using the equality generated by the check to align the types). In the above construction, this is always the case and we can establish this once and for all. Thus, the verification that Z_i is indeed a connector closely follows the provided proof sketch.

Theorem 3.4 (Menger's Theorem) *Let G be a digraph and let A, B be sets of vertices of G such that every AB-separator has at least size n. Then there exists an AB-connector of size at least n.*

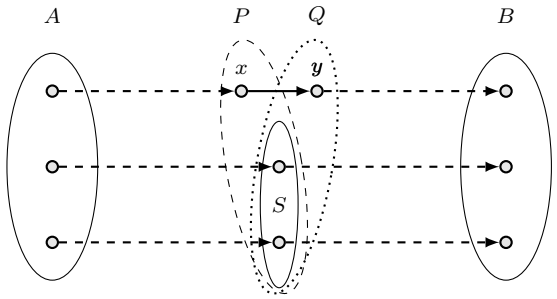


Fig. 1: Objects occuring the the proof of Menger's Theorem

Proof By induction on $|\{(x,y) : G \times G \mid x{-}y\}|$. If G has no edges, then $A \cap B$ (seen as a collection of single-vertex paths) is a sufficiently large AB-connector. Hence, we can assume there are vertices $x, y : G$ such that $x{-}y$. Let $G' := G - xy$. Without loss of generality, G' has an AB-separator S with $|S| < n$. [Otherwise, we obtain an AB-connector by induction.] Let $P := S \cup \{x\}$ and $Q := S \cup \{y\}$. Both P and Q are AB-separators of G. [Assume there was some AB-path π in G avoiding x or y. Then π cannot use the xy-edge, hence π is a G'-path and visits S.] Thus, $n = |P| = |Q| = |S|+1$, i.e., $x, y \notin S$ and the situation looks as depicted in Figure 1. Now, every AP-separator (or QB-separator) of G' is an AB-separator of G. [To see this, let T be an AP-separator of G' and assume some AB-path π in G avoiding T. Then π visits the AB-separator P, and therefore must use the xy-edge. Splitting π at x yields an AP-path in G' avoiding T. Contradiction.] Thus, every AP-separator (or QB-separator) of G' has size at least n. By induction hypothesis, we obtain an AP-connector X and a QB-connector such that $|X| = |Y| = n$. The required AB-connector is then obtained using Lemmas 3.2 and 3.3. □

The above proof is almost exactly the proof given by Göring [18], we merely add some additional elaborations (i.e., the sentences enclosed in "[...]" and the two lemmas). Several important theorems (e.g, Hall's Marriage Theorem and Kőnig's Theorem) can be obtained as simple consequences of Menger's Theorem. For additional detail, we refer to [9] or the accompanying Coq development. In this paper we will only make use of a variant of Menger's Theorem establishing the existence of n independent paths between a pair of vertices x and y provided one needs to remove at least n vertices to disconnect x and y.

Definition 3.5 Let x, y be vertices of some digraph G. Two irredundant xy-paths π_1 and π_2 are *independent* if $\pi_1 \cap \pi_2 = \{x, y\}$. A set of vertices S *separates* x and y if $\{x, y\} \cap S = \emptyset$ and every xy-path contains a vertex from S.

Note that "S separates x and y" is a stronger statement than "S is a $\{x\}\,\{y\}$-separator". In particular, $\{x\}$ is always a $\{x\}\,\{y\}$-separator and never separates x and y.

Corollary 3.6 *Let G be a digraph, and let $x, y : G$ such that $x \neq y$ and there is no xy-edge. If $n \leq |S|$ for every set S separating x and y, then there exist n irredundant and pairwise independent xy-paths.*

Fig. 2: A graph of treewidth two (left) with tree decomposition (right)

Proof Let $G' := G|_{\overline{\{x,y\}}}$ by the subgraph of G induced by the complement of $\{x, y\}$. Let $A := \{z : G' \mid x{-}z\}$ and $B := \{z : G' \mid z{-}y\}$. Then every AB-separator of G' also separates x and y in G and therefore has size at least n. By Menger's Theorem, we obtain an AB-connector X of size n. Adding x at the start and y at the end of every path in X yields n independent xy-paths. □

4 Treewidth and Minors

We now define the notions of treewidth and minors. Both notions appear in the literature with slight (but equivalent) variations. We choose variants that yield reasonable proof principles.

Definition 4.1 A *forest* is a simple graph where there is at most one irredundant path between any two nodes.

In Coq, this is formalized by requiring that for every pair of vertices x and y, all irredundant xy-paths are equal.

Definition 4.2 Let G be a simple graph and let $A : 2^G$. A is a *clique* if $x{-}y$ whenever $\{x, y\} \subseteq A$ and $x \neq y$.

Definition 4.3 A *tree decomposition* of a simple graph G is a forest F together with a function $B : F \to 2^G$ such that:
T1. for every vertex $x : G$, there exists some $t : F$, such that $x \in B(t)$.
T2. for every $x : G$, the set $\{t : F \mid x \in B(t)\}$ is connected in F;
T3. if $x{-}y$, then there exists a node t, such that $\{x, y\} \subseteq B(t)$;
The *width* of a tree decomposition is the size of the largest set $B(t)$ minus one; the *treewidth* of a graph is the minimal width of a tree decomposition.

An example of a graph together with a tree decomposition certifying that the graph has treewidth at most two is given in Figure 2.

The minus one in the definition of treewidth is there solely to ensure that trees have treewidth one. In the formalization we take the width so be the size of the largest bag (without substracting one) and adapt the statements accordingly.

Note that we define the notion of tree decomposition using forests rather than trees as is done for instance in [8]. The two notions are equivalent since every forest can be turned into a tree by connecting arbitrary nodes of disconnected trees. Using forests rather than trees has the advantage that tree decompositions for the disjoint union of two graphs G and H can be obtained as the disjoint union of tree decompositions for G and H.

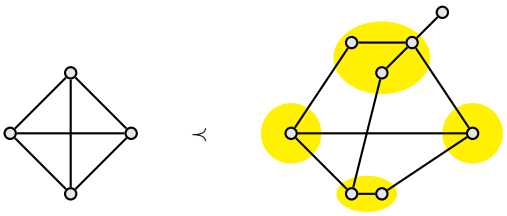


Fig. 3: Exhibiting K_4 as a minor

Definition 4.4 Let G and H be simple graphs. We write $G + H$ for the disjoint union of G and H (i.e., the graph with vertices $\mathsf{inl}\, x$ for $x : G$ and $\mathsf{inr}\, y$ for $y : H$ such that $\mathsf{inl}\, x$–$\mathsf{inl}\, y$ iff x–y in G and likewise for H.)

Lemma 4.5 *Let G_1 and G_2 be simple graphs, T_1 and T_2 forests, $B_1 : T_1 \to 2^{G_1}$ a tree decomposition of G_1, and $B_2 : T_2 \to 2^{G_2}$ a tree decomposition of G_2. Then*

$$\lambda u : T_1 + T_2. \begin{cases} B_1(x) & u = \mathsf{inl}\, x \\ B_2(x) & u = \mathsf{inr}\, x \end{cases}$$

is a tree decomposition of $G_1 + G_2$.

The minors of a graph G are customarily defined to be those graphs that can be obtained by a series of the following operations: remove a vertex, remove an edge, or contract an edge. We use instead a monolithic definition in terms of functions to sets inspired by [8].

Definition 4.6 Let G and H be simple graphs. A function $\phi : H \to 2^G$ is called a *minor map* if:
M1. $\phi(x)$ is nonempty and connected for all $x : H$,
M2. $\phi(x) \cap \phi(y) = \emptyset$ whenever $x \neq y$ for all $x, y : H$.
M3. $\phi(x)$ neighbors $\phi(y)$ for all $x, y : H$ such that x–y.
H is a *minor* of G, written $H \prec G$ if there exists a minor map $\phi : H \to 2^G$.

Intuitively, for every vertex $x : H$, $\phi(x)$ is the set of vertices being collapsed to x by contracting edges in $\phi(x)$. Consequently we will refer to $\phi(x)$ as the *inflation* of x.

In [11], we employed an equivalent definition using functions of type $G \to H_\perp$ instead of $H \to 2^G$ and expressing the conditions on the preimages. For the results in this paper, we found Definition 4.6 more convenient; the Coq development contains both definitions and establishes their equivalence.

Definition 4.7 We write K_4 for the complete graph with four vertices. A graph G is called K_4-*free*, if $\mathsf{K}_4 \not\prec G$.

As an example, consider the graph obtained by adding an edge between vertices 4 and 7 of the graph in Figure 2. This graph (depicted on the right of Figure 3) admits K_4 as minor by collapsing the circled sets of vertices. Note that there are in general many ways to contract a connected set to a single vertex and our definition abstracts from these unimportant variations.

Lemma 4.8 *If $\phi : G \to 2^H$ and $\psi : H \to 2^I$ are minor maps, then $\lambda x.\ \bigcup_{y \in \phi(x)} \psi(y)$ is a minor map of type $G \to 2^H$.*

As a consequence of the lemma above, we obtain that $\prec$ is transitive. Moreover, we have that the class of graphs with treewidth below a certain threshold is closed under taking minors.

Lemma 4.9 *If $H \prec G$, then the treewidth of H is at most the treewidth of G.*

Proof Let (T, B) be a tree decomposition of G and let $\phi : H \to 2^G$ be a minor map. Then $D(t) := \{x : H \mid \phi(x) \cap B(t) \neq \emptyset\}$ is a tree decomposition of H. Moreover, $\phi(x)$ and $\phi(y)$ are disjoint whenever $x \neq y$, so $|D(t)| \leq |B(t)|$. □

One of our main results is a formal proof that the K_4-free graphs are precisely those graphs that have tree decompositions of width at most two. One direction, showing that a graph cannot both have a tree decompositon of width at most two and K_4 as a minor, is relatively straightforward. It is an easy consequence of Lemma 4.9 and the theorem below. The converse direction, constructing low-width tree decompositions for K_4-free graphs is more involved; we defer the proof to Section 5.

Theorem 4.10 *Let T be a (nonempty) forest and let $B : T \to 2^G$ be a tree decomposition of G. Then every clique of G is contained in $B(t)$ for some $t : T$.*

Proof We prove by induction on n that every clique S with $|S| \leq n$ is contained in $B(t)$ for some t. If $|S| \leq 1$ this follows with (T1). Thus, we can assume $\{v_0, v\} \subseteq S$ with $v_0 \neq v$. We define

$$S_0 := S \setminus \{v\} \qquad\qquad T_0 := \{t : T \mid S_0 \subseteq B(t)\}$$

By induction hypothesis, T_0 is nonempty. Suppose by contradiction that $v \notin B(t)$ for all $t \in T_0$. Since S is a clique, there exists some $t : T$ such that $\{x, y\} \subseteq B(t)$ for every set $\{x, y\} \subseteq S$ (T3). Let c be such that $\{v, v_0\} \subseteq B(c)$ and let C be the component of c in $T \setminus T_0$, i.e.:

$$C := \{t : T \mid \exists \pi : c \rightsquigarrow t.\ \pi \cap T_0 = \emptyset\}$$

Then C is connected, disjoint from T_0 and contains c. Since v_0 occurs both in T_0 and in C, there exist $t_0 \in T_0$ and $c_0 \in C$ such that $t_0 - c_0$ (T2) and, since T is a forest, every path starting in C and ending in T_0 must use this edge. We obtain a contradiction by showing $c_0 \in T_0$.

Fix some $u \in S_0$; it suffices to show $u \in B(c_0)$. We have $u \in B(t_0)$. By (T3) we also have $\{u, v\} \subseteq B(c_u)$ for some c_u. Since $v \subseteq B(c) \cap B(c_u)$ the unique irredundant cc_u-path avoids T_0 (T2). Thus $c_u \in C$ and therefore $u \in B(c_0)$ since the unique t_0c_u-path must go through c_0 and (again by T2) every bag along the way must contain u. □

Corollary 4.11 *If G has treewidth at most two, then G is K_4-free.*

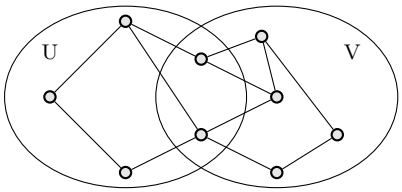


Fig. 4: Separation of a graph with a smallest separator of size two.

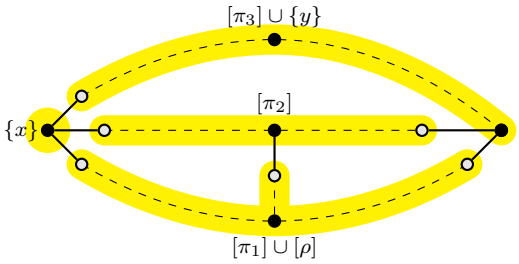


Fig. 5: Constructing K_4 from a separator of size three: The filled vertices are distinct, unfilled vertices may coincide with the filled vertices they are connected to by dashed lines.

5 Tree Decompositions for $\mathbf{K_4}$-free Graphs

We now prove the converse of Corollary 4.11, i.e. that every K_4-free graph has a tree decomposition of width at most two. For this, we need to construct an object (a low-width tree decomposition) from the knowledge that a certain substructure (the minor K_4) does not occur.

The proof we give is inspired by the argument in [8]; it is structured as follows: We first show that every sufficiently large K_4-free graph can be covered by two sets U and V whose intersection contains at most two vertices (cf. Figure 4). This part makes use of Menger's Theorem. We then show that if $U \cap V$ does contain two vertices, we can assume without loss of generality that there is an edge between them. This allows us to obtain a tree decomposition of G by obtaining tree decompositions of the subgraphs $G|_U$ and $G|_V$, identifying a bag containing $U \cap V$ in each of the decompositions (Theorem 4.10), and adding a link between those two bags. Arguing that $G+xy$ is still K_4-free in the case where $U \cap V = \{x, y\}$ but x and y are not adjacent (the nontrivial part of the without loss generality argument) requires analyzing a hypothetical minor map exhibiting K_4 as a minor of $G + xy$ and showing that this would yield $\mathsf{K}_4 \prec G$. This is the most technical part of the proof.

Definition 5.1 (Separators and Connectivity) Let G be a simple graph. A *separator* is a set S disconnecting G, i.e., separating two distinct vertices of G. The graph G is *k-connected* if $|G| > k$ and every separator has at least k vertices.

In order to prove the next lemma, we need an auxiliary notion. If π is an xy-path, we write $[\pi]$ for the set $\pi \setminus \{x, y\}$. Note that if π is irredundant, then $[\pi]$ is connected; if $[\pi]$ is also nonempty, then $[\pi]$ neighbors any set containing either x or y.

Lemma 5.2 *Every 3-connected graph includes* K_4 *as a minor.*

Proof Let G be 3-connected. If G is complete, we clearly have $\mathsf{K}_4 \prec G$. Otherwise let x and y be two vertices such that there is no xy-edge. By Corollary 3.6, we obtain three independent xy-paths π_1, π_2 and π_3. Moreover, all these paths have a non-empty interior. Since $\{x, y\}$ is not a separator, there exists a path ρ connecting, without loss of generality, $[\pi_1]$ and $[\pi_2]$ such that $[\rho] \cap (\pi_1 \cup \pi_2 \cup \pi_3) = \emptyset$. The minor map mapping the four vertices of K_4 to the following four sets establishes K_4 as a minor of G: $\{x\}$, $[\pi_1] \cup [\rho]$, $[\pi_2]$, and $[\pi_3] \cup \{y\}$ (cf. Figure 5). □

As an immediate consequence of the lemma above we obtain the existence of small separators in K_4-free graphs.

Lemma 5.3 *Let G be a simple K_4-free graph with at least four vertices. Then there exists a smallest separator S of G with $|S| \leq 2$.*

We want to view the separators obtained using the lemma above as separating the graph into exactly two components (cf. Figure 4). This motivates the notion of separation.

Definition 5.4 (Separation) Let $G = \langle V, R \rangle$ be a simple graph. A pair of sets (V_1, V_2) is a *separation* (of G) if $V_1 \cup V_2 = V$ and $\overline{V_2}$ and $\overline{V_1}$ are *not* neighboring.[3] A separation is *proper*, if both $\overline{V_1}$ and $\overline{V_2}$ are nonempty. The *order* of a separation is the size of $V_1 \cap V_2$. A *minimal* separation is a separation whose order is minimal.

The main difference between a separator and a proper separation is that the latter specifies for every vertex which "side" of the separator the vertex is on.

Lemma 5.5 *1. If (V_1, V_2) is a separation, then $V_1 \cap V_2$ separates all elements of $\overline{V_1}$ from all elements of $\overline{V_2}$. Thus, $V_1 \cap V_2$ is a separator whenever (V_1, V_2) is proper.*
2. For every separator S, there exists a proper separation (V_1, V_2), with $V_1 \cap V_2 = S$.

Next, we show that a separation whose shared part is a clique allows us to combine tree decompositions for the two parts.

Lemma 5.6 *Let G be a graph and let (V_1, V_2) be a separation of G such that $V_1 \cap V_2$ is a clique. Further let (T_i, B_i) be a tree decomposition of $G|_{V_i}$ for $i \in \{1, 2\}$. Then there exists a tree decomposition of G of width $\mathsf{max}(\mathsf{width}(B_1), \mathsf{width}(B_2))$.*

Proof Since $S := V_1 \cap V_2$ is a clique, there exist tree nodes $t_i : T_i$ such that $S \subseteq B_i(t_i)$ for $i \in \{1, 2\}$ (Theorem 4.10). We obtain a tree decomposition of G by taking the disjoint union of the tree decompositions of $G|_{V_1}$ and $G|_{V_2}$ and then adding a $t_1 t_2$-edge. □

In order to use the aforementioned small separators to decompose graphs in such a way that their respective tree decompositions yield a tree decomposition of the full graph, we need to show that if $\{x, y\}$ is a smallest separator, then adding an xy-edge preserves K_4-freeness. We first establish two auxiliary lemmas.

Lemma 5.7 *Let $\mathsf{K}_4 \prec G$ and let (V_1, V_2) be a minimal separation of G such that $V_1 \cap V_2$ is a clique of size at most two. Then there exists a minor map ϕ such that either $\phi(x) \subseteq V_1$ for all $x : \mathsf{K}_4$ or $\phi(x) \subseteq V_2$ for all $x : \mathsf{K}_4$*

[3] Note that $\overline{V_2} = V_1 \setminus V_2$ if $V_1 \cup V_2 = V$.

Proof Let $G = \langle V, E\rangle$ and let $\phi : \mathsf{K}_4 \to 2^G$ be a minor map. We first show that, without loss of generality, we can assume $\phi(i) \cap V_1 \neq \emptyset$ for all i. To see this, consider i, j such that $\phi(i) \cap V_1 = \emptyset$ and $\phi(j) \cap V_2 = \emptyset$. If $i = j$, we have a contradiction since $\phi(i)$ is nonempty and $V_1 \cup V_2 = V$. Similarly, $i \neq j$ contradicts the assumption that $\overline{V_1}$ does not neighbor $\overline{V_2}$. Now, since (V_1, V_2) is a separation, every $\phi(i)$ containing a vertex from $\overline{V_1}$ must also contain a vertex from $S := V_1 \cap V_2$. Thus, $\phi'(i) := \phi(i) \cap V_1$ is a minor map as required (using the fact that S is a clique to ensure that the various $\phi(i)$ are connected and neighboring). □

Lemma 5.8 *If (V_1, V_2) is a minimal proper separation, then every vertex $x \in V_1 \cap V_2$ is adjacent to a vertex in $\overline{V_1}$ and a vertex in $\overline{V_2}$.*

The following lemma is the central construction of this section.

Lemma 5.9 *Let G be K_4-free and let $S = \{x, y\}$ (with $x \neq y$) be a smallest separator. Then $G + xy$ is K_4-free.*

Proof Assume $\mathsf{K}_4 \prec G + xy$ and let $\phi : \mathsf{K}_4 \to 2^{(G+xy)}$ be a minor map. Further, let (V_1, V_2) be a proper separation with $S = \{x, y\}$. By Lemma 5.7, we can assume, without loss of generality, that $\phi(i) \subseteq V_1$ for all i. Since G is K_4-free, the xy-edge must be used in one of two ways: (A) to ensure that $\phi(i)$ and $\phi(j)$ are neighboring (in $G + xy$) for some $\phi(i)$ and $\phi(j)$ not neighboring in G; or (B) to ensure that $\phi(i)$ is connected (in $G + xy$) for some $\phi(i)$ not connected in G.

Case A: Without loss of generality, we have $x \in \phi(i)$ and $y \in \phi(j)$. By Lemma 5.8, there exists a vertex $z \notin V_1$ such that x–y. Since $\{x\}$ is not a separator, we obtain an irredundant zy-path π avoiding x. Since S is a separator, $\pi \cap V_1 = \{y\}$. Thus, $\phi[j := \phi(j) \cup \pi] : \mathsf{K}_4 \to G$ is a minor map, contradicting the assumption that G is K_4-free.

Case B: Let $C(v)$ be the component of v in $\phi(i)$ (seen as a set in G). Then $\phi(i) = C(x) \cup C(y)$ and $C(x)$ and $C(y)$ are disjoint. If all $\phi(j)$ for $j \neq i$ are neighboring one of the two components (say $C(x)$), then $\phi[j := C(x)]$ is a minor map not using the xy-edge, again contradicting the K_4-freeness of G. Otherwise, we obtain without loss of generality some j such that $\phi(j)$ neighbors $C(x)$ while $\phi(k)$ neighbors $C(y)$ for $k \notin \{i, j\}$. Setting $\phi' := \phi[j := \phi(j) \cup C(x), i := C(y)]$ yields a minor map where the xy-edge is used to connect $\phi(i)$ and $\phi(j)$, reducing the problem to case A. □

Note that the monolithic definition of minor maps (Definition 4.6) allows for a straightforward analysis of how the xy-edge must be used by the minor map ϕ. Also note that the notion of neighboring sets is used pervasively throughout this section, allowing us to avoid having to explicitly exhibit edges in many cases. Putting everything together, we obtain:

Theorem 5.10 *Every K_4-free graph has a tree decomposition of width at most two.*

Proof By induction on $|G|$ for some K_4-free graph G. If $|G| \leq 3$, the claim is trivial; so assume $4 \leq |G|$. By Lemma 5.3, we obtain a smallest separator S such that $|S| \leq 2$. We can assume without loss of generality that S is a clique: if $S = \{x, y\}$ but $x \not\sim y$, then $G + xy$ is K_4-free by Lemma 5.9, and any tree decomposition for $G + xy$ is also a tree decomposition for G. We extend S into a proper separation (V_1, V_2) with $S = V_1 \cap V_2$ (Lemma 5.5). By induction hypothesis, we obtain tree decompositions for $G|_{V_1}$ and $G|_{V_2}$ of width at most two. Thus, we obtain the desired tree decomposition by Lemma 5.6. □

Note that Lemma 5.6 uniformly deals with the cases of disconnected graphs, graphs that are separated by a single vertex and graphs with separators of size two. In the latter case, Lemma 5.9 is needed to establish the clique condition. That is Lemma 5.9, the main construction underlying Theorem 5.10, is merely used to justify a "without loss of generality" step.

6 Labeled multigraphs

The initial motivation of this work was the characterisation of K_4-free graphs as the free 2p-algebra [22]. There, graphs have labeled edges, and parallel edges are allowed. We define *labeled multigraphs* accordingly, with labels in a fixed alphabet Σ.

Definition 6.1 A *(labeled directed) multigraph* is a structure $G = \langle V, E, s, t, l\rangle$, where V is a finite type of *vertices*, E is a finite type of *edges*, $s, t : E \to V$ are functions indicating the *source* and *target* of each edge, and $l : E \to \Sigma$ is function indicating the *label* of each edge. If G is a multigraph, we write $x : G$ to denote that x is a vertex of G.

Note that self-loops are allowed, as well as parallel edges with the same label.

This definition is rather different from the previous ones for directed and simple graphs: edges are represented explicitely as a (finite) type. This corresponds to the standard representation of graphs in category theory and this makes it possible to use the following notion of isomorphism, where the identity of edges is taken into account.

Definition 6.2 A *homomorphism* from the graph $G = \langle V, E, s, t, l\rangle$ to the graph $G' = \langle V', E', s', t', l'\rangle$ is a pair $\langle f, g\rangle$ of functions $f : V \to V'$ and $g : E \to E'$ that respect the various components: $s' \circ g \equiv f \circ s$, $t' \circ g \equiv f \circ t$, and $l \equiv l' \circ g$.

An *isomorphism* is a homomorphism whose two components are bijective functions. We write $G \simeq G'$ when there exists an isomorphism between graphs G and G'.

The corresponding Coq definitions require some care. Indeed, more than the existence of isomorphisms, we often need to keep track of their action on vertices and edges. This is typically the case in the following section, where we work with multigraphs with distinguished vertices. To this end, we define isomorphisms in Type rather than Prop, and we rely on the following notion of bijection between types, where the inverse function is given explicitly.

```
Record bij (A B: Type): Type := {
  fwd  :> A → B;
  bwd  :  B → A;
  bijK :  ∀ a, bwd (fwd a) = a;
  bijK' : ∀ b, fwd (bwd b) = b }.
```

```
Record iso (F G: graph): Type := {
  iso_v   :> bij (vertex F) (vertex G);
  iso_e   :  bij (edge F) (edge G);
  src_iso :  ∀ e, src (iso_e e) = iso_v (src e);
  tgt_iso :  ∀ e, tgt (iso_e e) = iso_v (tgt e);
  lbl_iso :  ∀ e, lbl (iso_e e) = lbl e }.
```

By declaring the first fields of those structures as coercions, we can freely use a bijection as a function, and an isomorphism as a function on vertices. In addition, we setup specific notations to use the inverse of a bijection, as well as the edge component of an isomorphism.

Even though these definitions are computational, it is convenient to think of them as equivalence relations on types and graphs. We use for that the extension of Coq' setoid rewriting tactics [30] to Type valued relations (a proof relevant extension inspired by homotopy type theory [32]).

Every labeled multigraph can be seen as a directed graph by forgetting labels as well as edge identities and multiplicities. By further forgetting edge directions and removing self-loops, we obtain a simple graph which we call the *skeleton*.

Definition 6.3 Let $G = \langle V, E, s, t, l\rangle$. The *skeleton* of $\langle V, E, s, t, l\rangle$ is the simple graph $\langle V, R\rangle$ where xRy iff $x \neq y$ and there exists an edge $e : E$ such that $s(e) = x$ and $t(e) = y$ or vice versa.

Skeletons allow us to reuse our definitions and results about simple graphs on multigraphs, e.g., those about minors and treewidth. That taking the skeleton of a graph does not change the type of vertices greatly simplifies lifting properties of the skeleton to the graph and vice versa. In practice, we turn the construction of taking the skeleton into a coercion from multigraphs to simple graphs.

In addition to comparing labeled multigraphs up to isomorphism, we will need to compose them in various ways to show that they form an algebra. This usually involves taking two graphs and merging (or identifying) some vertices in their disjoint union. To this end, we define the generic operations of taking the disjoint union of two graphs and quotienting a graph:

Definition 6.4 Let $G = \langle V, E, s, t, l\rangle$ and $G' = \langle V', E', s', t', l'\rangle$. The *disjoint union* of G and G', written $G + G'$, is defined to be the graph

$$\langle V + V', E + E', s + s', t + t', l + l'\rangle$$

Here, $f + f'$ (for $f \in \{s, t, l\}$) is the pointwise lifting of f and f' to the sum type $E + E'$ (with result in $V + V'$ or Σ).

Definition 6.5 Let $G = \langle V, E, s, t, l\rangle$ and let $\approx : G \to G \to \mathbb{B}$ be an equivalence relation. The *quotient of G modulo $\approx$*, written $G_{/\approx}$, is defined to be the graph

$$\langle V_{/\approx}, E, \pi \circ s, \pi \circ t, l\rangle$$

In order to prepare the ground for the following section, we prove several isomorphism lemmas about those operations. Here is a non-exhaustive list:

Lemma 6.6 *For all multigraphs F, F', G, G', H, we have:*

1. *$F + G \simeq G + F$ and $F + (G + H) \simeq (F + G) + H$.*
2. *If $F \simeq G$ and $F' \simeq G'$, then $F + F' \simeq G + G'$.*
3. *If $\approx$, $\approx'$ are two pointwise equivalent equivalence relations on (the vertices) of F, then $F_{/\approx} \simeq F_{/\approx'}$.*
4. *If $F \simeq G$ then $F_{/\approx} \simeq G_{/\approx'}$, where $\approx'$ is the equivalence relation on G induced through the given isomorphism by a given equivalence relation $\approx$ on F.*
5. *$F + G_{/\approx} \simeq (F + G)_{/\approx'}$ where $\approx$ is an equivalence relation on G and $\approx'$ is its extension to $F + G$ (leaving all vertices of F in singleton classes).*
6. *$\left(F_{/\approx}\right)_{/\approx'} \simeq F_{/\approx''}$, where $\approx$ is an equivalence relation on F, $\approx'$ is an equivalence relation on $F_{/\approx}$, and $\approx''$ is the equivalence relation on F obtained by composing $\approx$ and $\approx'$.*

7. $(F+G)_{/\approx} \simeq F_{/\approx'}$ *when* G *has no edge,* $\approx$ *is an equivalence relation on* $F+G$, $\approx'$ *is its restriction to* F, *and for all* $x : G$ *there exists* $y : F$ *with* $\mathsf{inr}\, x \approx \mathsf{inl}\, y$.

The first three items are straightforward. Item (2) makes it possible to rewrite isomorphims in contexts composed of disjoint unions; Item (4) extends this to rewriting isomorphisms under quotients. The latter illustrates the need to work with computational definitions: the induced equivalence relation $\approx'$ depends on the concrete behaviour of the given isomorphism. The last three items are more involved; they make it possible to move quotients out of disjoint unions and to compress them: this is convenient to obtain 'normal forms' for nested quotients (on disjoint unions) occurring in the laws to be proved in the following section.

The proofs for establishing the various points in the above lemma follow the same pattern: we first exhibit the appropriate bijections on the underlying types, and then show that they can be extended into graph isomorphisms.

7 The 2p-algebra of two-pointed multigraphs

Using the operations on multigraphs defined in the previous section, we can construct a 2p-algebra [22] whose elements are multigraphs. The syntax of 2p-algebras is as follows.

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

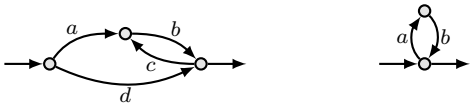
This syntax makes it possible to denote labeled multigraphs with two designated vertices (the *input* and the *output*).

Definition 7.1 A *two-pointed graph* (or *2p-graph* for short) is a structure $\langle G, \iota, o\rangle$ where G is a multigraph and $\iota : G$ and $o : G$ are two vertices called *input* and *output* respectively.

The notions of homomorphism and isomorphism are extended accordingly: on 2p-graphs, they should map the input to the input, and likewise for the outputs.

The algebra of 2p-graphs is defined in Figure 6, formally on the left, and informally on the right. The binary operations of the syntax correspond to series and parallel composition. We use the previous operations of disjoint union and quotient in their formal definition, where A^{eqv} denotes the equivalence relation generated by the pairs in A. The first unary operation, *converse*, exchanges input and output; the second one, *domain*, relocates the output to the input. The constant 1 represents the graph with just a single vertex; $\top$ is the disconnected graph with two vertices (one being the input and the other the output). Letters represent single edges.

This algebra allows us to recursively interpret every term t as a 2p-graph $\mathsf{g}(t)$. For instance, the graphs of the terms $a{\cdot}(b \parallel c^\circ) \parallel d$ and $1 \parallel a{\cdot}b$ are the following ones:



The second graph is also represented by the term $\mathrm{dom}(a \parallel b^\circ)$.

We remark that the definition of the function g works seemlessly in our setting where graphs are represented using finite types. More precisely, we make use of

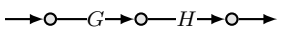
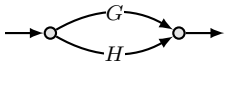
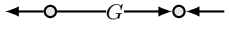
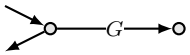
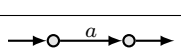
$\langle G,\iota,o\rangle{\cdot}\langle G',\iota',o'\rangle \triangleq \langle (G+G')_{/\approx}, \pi(\mathsf{inl}\,\iota), \pi(\mathsf{inr}\,o')\rangle$ where $\approx \triangleq \{(\mathsf{inl}\,o, \mathsf{inr}\,\iota')\}^{\mathsf{eqv}}$	
$\langle G,\iota,o\rangle \parallel \langle G',\iota',o'\rangle \triangleq \langle (G+G')_{/\approx}, \pi(\mathsf{inl}\,\iota), \pi(\mathsf{inl}\,o)\rangle$ where $\approx \triangleq \{(\mathsf{inl}\,\iota, \mathsf{inr}\,\iota'), (\mathsf{inl}\,o, \mathsf{inr}\,o')\}^{\mathsf{eqv}}$	
$\langle G,\iota,o\rangle^\circ \triangleq \langle G,o,\iota\rangle$	
$\mathrm{dom}(\langle G,\iota,o\rangle) \triangleq \langle G,\iota,\iota\rangle$	
$1 \triangleq \langle \langle \{*\}, \emptyset, \emptyset, \emptyset, \emptyset\rangle, *, *\rangle$	
$\top \triangleq \langle \langle \{0,1\}, \emptyset, \emptyset, \emptyset, \emptyset\rangle, 0, 1\rangle$	
$\underline{a} \triangleq \langle \langle \{0,1\}, \{*\}, \lambda\_.\,0, \lambda\_.\,1, \lambda\_.\,a\rangle, 0, 1\rangle$	

Fig. 6: The algebra of 2p-graphs.

the fact that the type of vertices is not part of the type of graphs and the fact that finite types are closed under disjoint union and quotients. Defining the function g in a setting where the vertices of graphs are given as subsets of some fixed type (as is the case in [26]) would require either a type closed under these operations (e.g., hereditarily finite sets as used by Paulson to formalize finite automata in Isabelle/HOL [27]) or an explicit encoding.

Two show that 2p-graphs form a 2p-algebra, we need to prove that the operations preserve isomorphisms, and satisfy the twelve axioms of 2p-algebras [22].

Lemma 7.2 *Let $F \simeq F'$ and $G \simeq G'$. Then we have $F \parallel G \simeq F' \parallel G'$, $F{\cdot}G \simeq F'{\cdot}G'$, and $F^\circ \simeq (F')^\circ$.*

Proof The first two points follow by using items 2 and 4 from Lemma 6.6: parallel and sequential composition are both defined as a quotient of a disjoint union. The third point is straightforward: the given isomorphism can be reused directly. □

Lemma 7.3 *For all 2p-graphs F, G, H, we have*

1. $F \parallel G \simeq G \parallel F$;
2. $(F \parallel G) \parallel H \simeq F \parallel (G \parallel H)$;
3. $(F{\cdot}G){\cdot}H \simeq F{\cdot}(G{\cdot}H)$;
4. $F{\cdot}1 \simeq F$;
5. $F^{\circ\circ} \simeq F$;
6. $(F \parallel G)^\circ \simeq F^\circ \parallel G^\circ$;
7. $(F{\cdot}G)^\circ \simeq G^\circ{\cdot}F^\circ$;
8. $\mathrm{dom}(F) \simeq 1 \parallel F{\cdot}\top$;
9. $1 \parallel 1 \simeq 1$;
10. $1 \parallel F{\cdot}G \simeq \mathrm{dom}(F \parallel G^\circ)$;
11. $F{\cdot}\top \simeq \mathrm{dom}(F){\cdot}\top$;
12. $(F \parallel 1){\cdot}G \simeq (F \parallel 1){\cdot}\top \parallel G$;

(Those laws are equivalent to those of 2p-algebras [22] and slightly easier to prove; that $\top$ is a neutral element for parallel composition follows from items 9 and 12.)

Proof All laws are easily checked informally by drawing pictures. Formalizing them however requires some non-trivial work. In particular, laws like associativity of parallel composition (2) do not follow directly from Lemma 6.6(1): parallel and sequential compositions involve quotients in addition to disjoint unions.

For most laws, we use Lemma 6.6(4,5,6) in order to rewrite both sides into a single quotient of a disjoint union of the starting graphs. Then we use Lemma 6.6(1,2,4,7) to align the disjoint unions and prune useless components (those appear when the starting expressions contain the constants 1 or $\top$). We finally conclude by using Lemma 6.6(3) and comparing the obtained equivalence relations.

For item (8) for instance, ignoring the inputs and outputs, the multigraph in the right-hand side is successively rewritten into expresions of the form $(1+(F+2))_{/\approx}$, $(F+(1+2))_{/\approx'}$, and $F_{/\approx''}$; it then suffices to show that $\approx''$ is the discrete equivalence relation. The equivalence relations used to define parallel and sequential compositions are easily described as the equivalence closures of short lists of pairs; we use variants of Lemma 6.6(4-7) maintaining this explicit representation so that it is straightforward to compare the final equivalence relations. □

Putting everything together, we obtain:

Theorem 7.4 *The set of 2p-graphs form a 2p-algebra (up to isomorphism).*

While the set of all 2p-graphs form a 2p-algebra, we need to restrict to graphs of treewidth at most two to obtain the free 2p-algebra. We show that those form a subalgebra, and in particular that the graph of every term has treewidth at most two. To be more precise, we need to restrict to 2p-graphs whose *strong skeleton* has treewidth at most two:

Definition 7.5 The *(weak) skeleton* of a 2p-graph $\langle G, \iota, o\rangle$ is the skeleton of G; its *strong skeleton* of is the skeleton of G with an additional ιo-edge.

The following lemma makes it possible to show that both series and parallel composition preserve treewidth two.

Lemma 7.6 *Let $G_1 = \langle G_1', \iota, o\rangle$ and $G_2 = \langle G_2', \iota', o'\rangle$ be 2p-graphs and let $\langle T_i, B_i\rangle$ ($i \in \{1,2\}$) be tree decompositions of the strong skeletons of G_1 and G_2 respectively. Further let $\approx$ be an equivalence relation on $G_1 + G_2$ identifying at least two vertices from the set $P \triangleq \{\mathsf{inl}\,\iota, \mathsf{inr}\,\iota', \mathsf{inl}\,o, \mathsf{inr}\,o'\}$ and no other vertices. Then there exists a tree decomposition of the skeleton of $(G1 + G2)_{/\approx}$ of width at most two having a node t such that $P_{/\approx} \subseteq B(t)$.*

Proof We use the three following facts. 1) A tree decomposition for a disjoint union of simple graphs can be obtained by taking the disjoint union of tree decompositions for those graphs. 2) Two trees of a tree decomposition can be joined through a new node containing the vertices of its neighbors. 3) A tree decomposition can be quotiented (to give a tree decomposition of a quotiented graph) as soon as it has nodes for all equivalence classes. □

Theorem 7.7 *For every term u, the strong skeleton of $\mathsf{g}(u)$ has a tree decomposition of width at most two.*

Proof By induction on u. The cases for $\|$ and $\cdot$ follow with Lemma 7.6. All other cases are trivial. □

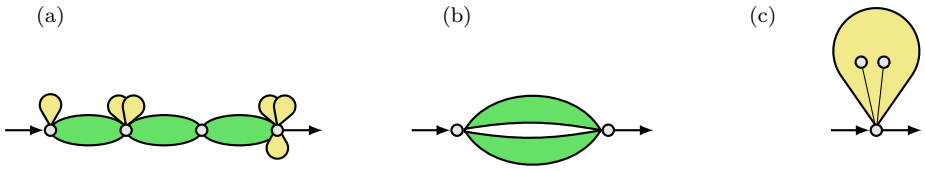


Fig. 7: The three main cases for extracting a term from a K_4-free graph.

8 Checkpoints

In the following two sections we show that Theorem 7.7 is sharp in the sense that all graphs whose strong skeleton has treewidth two can, up to isomorphism, be constructed as $\mathsf{g}(u)$ for some term u. We prove this by defining an *extraction* function taking as input some K_4-free graph and returning a term describing the graph. (Such a function is not unique since different terms may denote the same graph, and there are several ways of defining such a function. In particular a natural and possibly simpler idea would be to proceed by induction on the tree decomposition. We follow the approach from [22] because the resulting function makes it possible to obtain (†); alternatively, we could use the rewriting system from [12].)

We focus on connected graphs first: they form a subalgebra when removing $\top$ from the signature, and they can be decomposed recursively; we extend the extraction function to handle all graphs in a second step (Section 10). Before we can define the extraction function, we need a number of results on simple graphs. These will allow us to analyze the structure of 2p-graphs (via their skeletons), facilitating the termination and correctness arguments for the extraction function.

We use the concept of *checkpoint* to extract terms from graphs; those are the vertices which every path between input and output must visit. Using those, we get that every connected graph with distinct input and output has the shape depicted in Figure 7(a), where the checkpoints are the only depicted vertices. One can parse such a graph as a sequential composition and proceed recursively once we have proved that the strong skeletons of the green and yellow components are K_4-free whenever this is the case for the starting graph.

If there are no proper checkpoints between input and output, we exploit a key property of K_4-free graphs: in such a case, either the graph is just an edge or it consists of at least two parallel components, making it possible to proceed recursively (cf. Figure 7(b)).

The last case to consider is when the input and the output of the graph coincide. Then the graph either consists only of a single vertex (possibly with self loops) or one can recursively extract a term for the graph obtained by relocating the output to one of the neighbors of the input and use the domain operation to recover the starting graph (cf. Figure 7(c)).

For the remainder of this section, G refers to some *connected* simple graph.

Definition 8.1 The *checkpoints* between two vertices x, y are the vertices which every xy-path must visit:

$$\mathsf{cp}\, x\, y \triangleq \{z \mid \text{every } xy\text{-path crosses } z\}$$

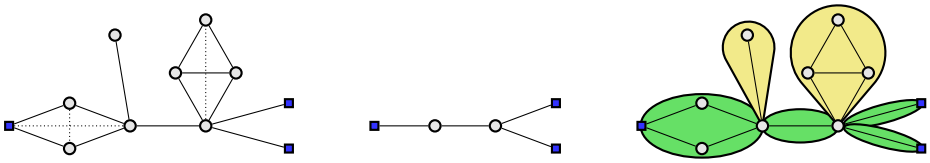


Fig. 8: Link graph, checkpoint graph, and decomposition into intervals and bags.

A checkpoint $z \in \mathsf{cp}\,x\,y$ is called *proper* if $z \notin \{x, y\}$. Two vertices x, y are *linked*, written $x \Diamond y$, when $x \neq y$ and $\mathsf{cp}\,x\,y = \{x, y\}$, i.e., when there are no proper checkpoints between x and y. The *link graph* of G is the graph of linked vertices.

Note that the link graph of G includes all the edges of G. Consider the graph on the left in Figure 8; its link graph is obtained by adding the three dotted edges to the existing ones.

Fact 8.2 *Let $x, y, z : G$. Then z is a proper checkpoint between x and y iff $\{z\}$ separates x and y.*

Lemma 8.3 *1.* $\mathsf{cp}\,x\,x = \{x\}$
2. $\{x, y\} \subseteq \mathsf{cp}\,x\,y = \mathsf{cp}\,y\,x$

Definition 8.4 Let U be a set of vertices of G. The *checkpoints* of U, written $\mathsf{CP}\,U$, are the vertices which are checkpoints of some pair in U.

$$\mathsf{CP}\,U \triangleq \bigcup_{x,y \in U} \mathsf{cp}\,x\,y$$

The *checkpoint graph* of U is the subgraph of the link graph induced by this set. We also denote this graph by $\mathsf{CP}\,U$.

The graph in the middle of Figure 8 is the checkpoint graph of the one of the left, when U consists of the blue square vertices.

Lemma 8.5 *Let $x, y \in \mathsf{CP}\,U$. Then $\mathsf{cp}\,x\,y \subseteq \mathsf{CP}\,U$.*

Proof We have $x \in \mathsf{cp}\,x_1\,x_2$ and $y \in \mathsf{cp}\,y_1\,y_2$ for some vertices $\{x_1, x_2, y_1, y_2\} \subseteq U$ by the definition of CP. Fix some $z \in \mathsf{cp}\,x\,y$. If $z \in \{x, y\}$, the claim is trivial, so assume $z \notin \{x, y\}$. Hence, we obtain either an xx_1-path or an xx_2-path not containing z by splitting some irredundant x_1x_2-path at x. Without loss of generality, the xx_1-path avoids z. Similarly, we obtain, again w.l.o.g., a yy_1-path avoiding z. Thus $z \in \mathsf{cp}\,x_1\,y_1$ since the existence of an x_1y_1-path avoiding z would contradict $z \in \mathsf{cp}\,x\,y$ (by concatenation with the paths obtained above). □

Definition 8.6 Let $x, y : G$. The *strict interval* $]\!]x; y[\!]$ is the following set of vertices.

$$]\!]x; y[\!] \triangleq \{p \mid \text{there is an } xp\text{-path avoiding } y \text{ and a } py\text{-path avoiding } x\}$$

The *interval* $[\![x; y]\!]$ is obtained by adding x and y to that set. We abuse notation and also write $[\![x; y]\!]$ for the subgraph of G induced by the set $[\![x; y]\!]$.

Definition 8.7 The *bag* of a checkpoint $x \in \mathsf{CP}\,U$ is the set of vertices that need to cross x in order to reach the other checkpoints.

$$[\![x]\!]_U \triangleq \{p \mid \forall y \in \mathsf{CP}\,U.\ \text{every } py\text{-path crosses } x\}\,.$$

As before, we also write $[\![x]\!]_U$ for the induced subgraph of G.

Note that $[\![x]\!]_U$ depends on U and differs from $[\![x;x]\!]$ (which is always the singleton $\{x\}$). The main purpose of bags and intervals is to aid in decomposing graphs for the term extraction function, as depicted on the right in Figure 8: the green components are intervals and the yellow components are bags. We first show that distinct bags, adjacent bags, and strict intervals are disjoint.

Lemma 8.8 *1. If $y \in \mathsf{CP}\,U$, then $[\![x]\!]_U \cap]\!]x;y[\![= \emptyset$.*
2. If $x, y \in \mathsf{CP}\,U$ and $x \neq y$, then $[\![x]\!]_U \cap [\![y]\!]_U = \emptyset$.
3. If $z \in \mathsf{cp}\,x\,y$, then $[\![x;y]\!] = [\![x;z]\!] \cup [\![z]\!]_{\{x,y\}} \cup [\![z;y]\!]$.
4. If $z \in \mathsf{cp}\,x\,y$, then $]\!]x;z[\![$, $[\![z]\!]_{\{x,y\}}$ and $]\!]z;y[\![$ are pairwise disjoint.

The following proposition is crucial for term extraction; it allows us to split graphs into parallel components provided the graph cannot be split into to sequential components (case (b) in Figure 7; this is the only place in the extraction where K_4-freeness of the input graph is being used rather than just being transferred to subgraphs).

Lemma 8.9 *Let $\iota, o : G$ such that $G + \iota o$ is K_4-free and $\iota \diamondsuit o$, but not $\iota - o$. Then $]\!]\iota;o[\![$ has at least two connected components.*

Proof Since $\iota \diamondsuit o$, we have that every set S separating ι and o has size ≥ 2. By Corollary 3.6, we obtain two independent ιo-paths π_1 and π_2 with nonempty interior. Fix $x_1 \in [\pi_i]$ and $x_2 \in [\pi_2]$. It suffices to show that x_1 and x_2 are not connected in $]\!]\iota;o[\![$. Assume there exists an x_1x_2-path ρ in $]\!]\iota;o[\![$. Let z be the first vertex on ρ that is in $[\pi_2]$ and let z' be the last vertex before z. Taking π_3 to be a one-edge ιo-path, we obtain the same situation as in Figure 5 (except that $[\pi_3] = \emptyset$, which doesn't influence the proof), so we reuse the construction underlying Lemma 5.2 □

Lemma 8.9 corresponds to [22, Proposition 20(i)], where it is proved by building on a long series of additional lemmas about checkpoints; this is also the approach followed in [11]. Thanks to Corollary 3.6 of Menger's theorem, which we used to prove the minor exclusion theorem for treewidth at most two (Theorem 5.10), these lemmas are no longer required in the present formalisation. This should not come as a surprise since thanks to Theorem 7.7, the extraction function we are defining eventually leads to an alternative proof of Theorem 5.10. In a sense, Corollary 3.6 captures the central graph-theoretic argument.

In order to apply Lemma 8.9 to the various intervals in the sequential decomposition of a 2p-graph, we need to show that the strong skeletons induced by these intervals are again K_4-free.

Lemma 8.10 *Let $\iota, o : G$ such that $G + \iota o$ is K_4-free and let $x, y \in \mathsf{cp}\,\iota\,o$ such that $x \neq y$. Then $[\![x;y]\!] + xy$ is K_4-free.*

Proof Without loss of generality x appears before y on every ιo-path. We obtain that $[\![x;y]\!] + xy$ is a minor of $G + \iota o$ by collapsing $[\![x]\!]_{\{x,y\}}$ (which contains ι) to x and $[\![y]\!]_{\{x,y\}}$ (which contains o) to y. The claim then follows by transitivity of the minor relation. □

9 Extracting Terms from $\mathsf{K_4}$-free Graphs

We say that a 2p-graph G is *CK4F* if its skeleton is connected and its strong skeleton is $\mathsf{K_4}$-free. We now define a function extracting terms from CK4F graphs. Defining this function in Coq is challenging for a number of reasons. First, its definition involves ten cases, most with multiple recursive calls. Second, we need to argue that all the recursive calls are made on smaller graphs which are CK4F.

To facilitate the definition, we construct our own operator for bounded recursion. The reason for this is that none of the facilities for defining functions in Coq (e.g., Fixpoint, Function and Program) are suited to deal with the kind of complex function definition we require. We define a bounded recursion operator with the following type:

```
Fix : ∀ aT rT : Type, rT → (aT → ℕ) → ((aT → rT) → aT → rT) → aT → rT
```

Here the argument of type $\mathsf{aT} \to \mathbb{N}$ is a measure on the input to bound the number of recursive calls, and the argument of type rT is the default value to be returned when no more recursive calls are allowed.

We only need one lemma about the recursion operator, namely that the operator satisfies the usual fixpoint equation provided that the functional it is applied to calls its argument only on smaller arguments in the desired domain of the function (here, CK4F).[4] That is, we have the following lemma:

```
Fix_eq : ∀ (aT rT : Type) (P : aT → Prop) (x0 : rT) (m : aT → ℕ)
  (F : (aT → rT) → aT → rT),
  (∀ (f g : aT → rT) (x : aT),
    P x → (∀ y : aT, P y → m y < m x → f y = g y) → F f x = F g x) →
  ∀ x : aT, P x → Fix x0 m F x = F (Fix x0 m F) x
```

While its proof is straightforward, this lemma is useful in that it allows us to abstract from the fact that we are using bounded recursion (i.e., neither the default result nor the recursion on $\mathbb{N}$ are visible in the proofs).

We now define the extraction function using the recursion operator. The various cases of the definition roughly correspond to the cases outlined in Figure 7. The main difference is that in case (a), rather than partitioning the graph as shown in the picture, we only identify a single nontrivial bag or a single proper checkpoint between input and output. This is sufficient to make recursive calls on smaller graphs. In the case where input and output coincide (case (c)), we relocate the output and proceed recursively. This requires a measure that treats graphs with shared input and output as larger than those with distinct input and output:

Definition 9.1 Let $G = \langle\langle V, E, s, t, l\rangle, \iota, o\rangle$ be a 2p-graph. The *measure* of G is $2|E|$ if $\iota \neq o$ and $2|E| + 1$ if $\iota = o$.

The term extraction function is then defined as

$$\mathsf{t} \triangleq \mathsf{Fix}\,1\,\mathsf{measure}\,\mathsf{F}$$

where the definition of F is given in Figure 9. This definition makes use of a number of auxiliary constructions which we define below. For a set of vertices U and a set of edges E (of some graph G) such that $\{s(e), t(e)\} \subseteq U$ for all e, the *subgraph of* G

[4] To be precise, F may call its argument on anything. However, the result of F may only depend on calls to smaller arguments in the domain.

```
1: Definition F(t : 2p-graph → term)(G : 2p-graph) ≜
2:     let ⟨⟨V, E′, s, t, l⟩, ι, o⟩ := G in
3:     if ι = o then
4:         let E := E({ι}) in
5:         if E = ∅ then
6:             if pick(components (V \ {i})) is Some C then
7:                 dom(t(redirect C)) ∥ t(G[C])
8:             else 1
9:         else (* E ≠ ∅ *)
10:            (||_{e∈E} tm(e)) ∥ G[V, E]
11:    else (* i ≠ o *)
12:        if E([ι]_{ι,o}) = ∅ ∧ E([o]_{ι,o}) = ∅ ∧ cp ι o = {ι, o} then
13:            let P := components (]ι; o[) in
14:            let E := E({ι, o}) in
15:            if E = ∅ then
16:                if pick P is Some C then
17:                    t(component(C)) ∥ t(G[C])
18:                else 1 (* never reached *)
19:            else (* E ≠ ∅ *)
20:                if P = ∅ then
21:                    ||_{e∈E} tm(e)
22:                else
23:                    (||_{e∈E} tm(e)) ∥ t(G[V, E])
24:        else (* nontrivial ι or o-bag or proper checkpoint between ι and o *)
25:            if E([ι]_{ι,o}) ≠ ∅ ∨ E([o]_{ι,o}) ≠ ∅ then
26:                t(G[ι])·t(G[ι, o])·t(G[o])
27:            else
28:                if pick (cp ι o \ {ι, o}) is Some z then
29:                    t(G[ι, z])·t(G[z])·t(G[z, o])
30:                else 1 (* never reached *)
```

Fig. 9: The term extraction function

with vertices U and edges E is written $G[U, E]$. We write $\mathcal{E}(U)$ for the set of edges with source and target in U and the *induced subgraph for* U, written $G[U]$, is defined as $G[U, \mathcal{E}(U)]$. For 2p-graphs G, $G[U]$ and $G[U, E]$ are only defined if $\{\iota, o\} \subseteq U$. In this case, $G[U]$ and $G[U, E]$ have the same input and output as G.

When instantiating the definitions above, U will sometimes be an interval or a bag. In this case, the intervals and bags are computed on the weak skeleton of G (not the strong skeleton). For a given 2p-graph $G = \langle G', \iota, o\rangle$, we also define:

$$
\begin{aligned}
\mathsf{components}(U) &\triangleq \{C \mid C \text{ connected component of } U \text{ in the skeleton of } G\}\\
\mathsf{component}(C) &\triangleq G[C \cup \{\iota, o\}]\\
\mathsf{redirect}(C) &\triangleq \langle G'[C \cup \{\iota\}], \iota, x\rangle \text{ where } x \text{ is some neighbor of } \iota \text{ in } C\\
G[x, y] &\triangleq \langle G'[[\![x; y]\!], \mathcal{E}([\![x; y]\!]) \setminus (\mathcal{E}(\{x\}) \cup \mathcal{E}(\{y\}))], x, y\rangle\\
G[x] &\triangleq \langle G'[[\![x]\!]_{\{\iota,o\}}], x, x\rangle\\
\mathsf{tm}(e) &\triangleq \begin{cases} l(e) & s(e) = \iota \wedge t(e) = o\\ l(e)^\circ & \text{otherwise}\end{cases}
\end{aligned}
$$

Note that $\mathsf{component}(C)$ is obtained as an induced subgraph of G whereas the other constructions are obtained as subgraphs of G' (with new inputs and outputs).

Before we can establish properties of t, we need to establish that all (relevant) calls to t in F are made on CK4F graphs with smaller measure.

Lemma 9.2 *Let t, t' be functions from graphs to terms. If t and t' agree on all CK4F graphs with measure smaller than a CK4F graph G, then $\mathsf{F}\,t\,G = \mathsf{F}\,t'\,G$.*

The proof of this lemma boils down to a number of lemmas for the various branches of F. For each recursive call, we need to establish both that the measure decreases and that the graph is indeed CK4F. When splitting of a parallel component (line 17), Lemma 8.9 ensures that there are at least two nonempty components, thus ensuring that the remainder of the graph is both smaller and connected. Note that the case distinction in line 20 is required since if $P = \emptyset$, removing the ιo-edges disconnects the graph (the remaining graph would be isomorphic to $\top$). In the case where there is a proper checkpoint z between input and output (line 29), Lemma 8.10 ensures that the strong skeletons of $G[\iota, z]$ and $G[z, o]$ are $\mathsf{K_4}$-free.

As a consequence of Lemma 9.2, we obtain:

Lemma 9.3 *Let G be CK4F. Then $\mathsf{t}\,G = \mathsf{F}\,\mathsf{t}\,G$.*

We finally show that interpreting the term extracted from a 2p-graph G yields a graph that is isomorphic to G. Together with the formal proof of Thm 7.4, this is where the difference of what one would find in a detailed paper proof and what is required in order to obtain a formal proof is greatest: the various isomorphisms have to be presented explicitely in the formal proof while they are left to the reader in a paper proof.

The extraction function decomposes the graph into smaller graphs in order to extract a term. The interpretation of this term then joins the graphs extracted by the recursive calls back together using the graph operations $\parallel$ and $\cdot$. We need to establish that the decomposition performed during extraction is indeed correct (i.e., that no vertices or edges are lost or misplaced). This requires establishing a number of additional isomorphism properties.

Among others, we establish the following isomorphism lemmas:

Lemma 9.4 *Let $G = \langle G', \iota, o\rangle$ such that $\iota \neq o$ and the skeleton of G is connected. Then $G \simeq G[\iota]{\cdot}G[\iota, o]{\cdot}G[o]$.*

Lemma 9.5 *Let $G = \langle G', \iota, o\rangle$ such that $\mathcal{E}(\llbracket \iota \rrbracket_{\{\iota,o\}}) = \emptyset$, $\mathcal{E}(\llbracket o \rrbracket_{\{\iota,o\}}) = \emptyset$, and $\iota \neq o$, and let $z \in \mathsf{cp}\,\iota\,o \setminus \{\iota, o\}$. Then $G \simeq G[\iota, z]{\cdot}G[z]{\cdot}G[z, o]$.*

Lemma 9.6 *Let $G = \langle G', \iota, o\rangle$ with $\mathcal{E}(\{\iota, o\}) = \emptyset$ and let $C \in \mathsf{components}(\overline{\{\iota, o\}})$. Then $G \simeq \mathsf{component}(C) \parallel G[\overline{C}]$.*

For the following, let $E_{x,y} \triangleq \{e \mid s(e) = x, t(e) = y\}$.

Lemma 9.7 *Let $G = \langle V, E, s, t, l\rangle$, let $x, y : G$ and let $E' \triangleq E_{x,y} \cup E_{y,x}$ Then $G \simeq G[\{x, y\}, E'] \parallel G[V, \overline{E'}]$.*

Theorem 9.8 *Let G be CK4F. Then $\mathsf{g}(\mathsf{t}\,G) \simeq G$.*

Proof By induction on the measure of G. We use Lemma 9.3 to unfold the definition of t. Each of the cases follows with the induction hypothesis (using the lemmas underlying the proof of Lemma 9.2 to justify that that the induction hypothesis applies) and some isomorphism lemmas (e.g., Lemmas 9.4 to 9.7). □

Note that Lemma 9.6 justifies both the split in line 7 and the split in line 17 (in the latter case $\llbracket \iota; o \rrbracket = \overline{\{\iota, o\}}$).

10 Handling disconnected graphs

Once we have obtained terms for graphs that are CK4F, it is straightforward to extend this result to (possibly) disconnected graphs. We define a function $\mathsf{t}^\top$ extracting terms from arbitrary 2p-graphs with K_4-free strong skeleton.

Definition 10.1 If $x : G$, we write G_x for the subgraph induced by the connected component of x (in G) with input and output set to x. We slightly abuse notation and also write G_x for the underlying set of vertices of G. We then define:

$$\mathsf{t}^\top(G) := \begin{cases} \top \cdot \mathsf{t}(G_x) \cdot \top \parallel \mathsf{t}^\top(G[\overline{G_x}]) & x \notin G_\iota \cup G_o \\ \mathsf{t}(G_\iota) \cdot \top \cdot \mathsf{t}(G_o) & \iota \text{ and } o \text{ disconnected} \\ \mathsf{t}(G) & \text{otherwise} \end{cases}$$

The three cases are to be read as ordered. That is, if there exists some $x \notin G_\iota \cup G_o$ we arbitrarily choose one; if not, we check whether ι and o disconnected and make the appropriate calls to t. The function $\mathsf{t}^\top$ terminates since $G[\overline{G_x}]$ has fewer vertices than G.

Theorem 10.2 *Let G be a 2p-graph with K_4-free strong skeleton. Then $\mathsf{g}(\mathsf{t}^\top(G)) \simeq G$.*

Proof By induction on $|G|$. If G has some component C containing neither ι nor o, it suffices to show $\top \cdot G[C] \cdot \top \parallel C[\overline{C}] \simeq G$, which follows with Lemma 7.3. Similarly, we have $G_\iota \cdot \top \cdot G_o \simeq G$ whenever G_ι and G_o are distinct and there are no other components. □

Recall that $\mathsf{g}(u)$ has treewidth at most two for all terms u (Theorem 7.7). Hence, Theorem 10.2 provides an alternative proof of Theorem 5.10. Note however, that the two theorems share essential constructions. As noted in Section 8, Menger's Theorem and the construction of K_4 from three independent paths (Figure 5) are shared between the two results (cf. Lemmas 5.2 and 8.9).

11 Directions for future work

We presented a library for graph theory built using Coq and the mathematical components library. The library contains formalizations of several interdependent results: Menger's Theorem [23,18], the excluded-minor characterisation for treewidth two [13], and a characterization of K_4-free 2p-graphs through the syntax of 2p-algebra [22,12].

Concerning the completeness of the axioms of 2p-algebras for isomorphism of 2p-graphs (†) [22], there is still substantial work to be done. Formalising the proof from [22] was our initial motivation, and while developing the formalization we found various simplifications. As described here, we have found that we can use Menger's Theorem to simplify the analysis of checkpoint graphs. However, we also found a completely different proof for the completeness of the axioms [12]. In [12], extraction of terms from graphs is done using a terminating and confluent rewrite system on term-labeled 2p-graphs. This approach is much simpler on paper. In particular, it does not reprove the excluded-minor characterization for treewidth-two. However, formalizing it brings its own challenges: we need to find an efficient

way of representing the rewriting system, whose confluence needs to be established in order to obtain completeness of the axioms.

The library keeps evolving, and many results from graph theory would be interesting to formalise as a next step. The excluded-minor characterizations of planar graphs [21] and outer-planar graphs [8] are natural candidates.

A different direction would be to use the library for the verification of graph algorithms. For instance, we do not currently provide functions for computing the treewidth of a graph or to check for graph isomorphisms. In the same vein, it would be nice to develop certified implementations of standard graph algorithms. It should be noted that the definitions in the library are geared towards declarative proofs and not towards the extraction of efficient code. We envision verifying algorithms in this abstract setting and then applying algorithmic and data refinements along the lines of [5].

Acknowledgements. We would like to thank Guillaume Combette, with whom we developped the first version of the library. We are also grateful to Nicolas Trotignon for his wonderful insights on graph theory.

References

1. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.
2. C. Chou. A formal theory of undirected graphs in higher-order logic. In *Proc. TPHOL*, volume 859 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1994.
3. C.-T. Chou. Mechanical verification of distributed algorithms in higher-order logic*. *The Computer Journal*, 38(2):152–161, 01 1995.
4. C. Cohen. Pragmatic quotient types in Coq. In *Proc. ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2013.
5. C. Cohen, M. Dénès, and A. Mörtberg. Refinements for free! In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs (CPP 2013)*, volume 8307 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2013.
6. B. Courcelle. The monadic second-order logic of graphs. I: Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
7. B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.
8. R. Diestel. *Graph Theory (2nd edition)*. Graduate Texts in Mathematics. Springer, 2000.
9. C. Doczkal. Short proof of Menger's Theorem in Coq (Proof Pearl). Working paper or preprint, Apr. 2019.
10. C. Doczkal, G. Combette, and D. Pous. Coq formalization accompanying this paper. `https://perso.ens-lyon.fr/damien.pous/covece/graphs/`.
11. C. Doczkal, G. Combette, and D. Pous. A formal proof of the minor-exclusion property for treewidth-two graphs. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving (ITP 2018)*, volume 10895 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2018.
12. C. Doczkal and D. Pous. Treewidth-two graphs as a free algebra. In *Proc. MFCS*, volume 117 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
13. R. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.
14. J. Dufourd and Y. Bertot. Formal study of plane Delaunay triangulation. In *Proc. ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 211–226. Springer, 2010.
15. E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. NCAI*, pages 4–9. AAAI Press / The MIT Press, 1990.
16. P. Freyd and A. Scedrov. *Categories, Allegories*. North Holland. Elsevier, 1990.

17. G. Gonthier. Formal proof — the four-color theorem. *Notices Amer. Math. Soc.*, 55(11):1382–1393, 2008.
18. F. Göring. Short proof of menger's theorem. *Discrete Mathematics*, 219(1-3):295–296, 2000.
19. M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1:1–1:24, 2007.
20. P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.
21. K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930. in French.
22. E. C. Llópez and D. Pous. K4-free graphs as a free algebra. In *Proc. MFCS*, volume 83 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
23. K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, pages 96–115, 1927.
24. Y. Nakamura and P. Rudnicki. Euler circuits and paths. *Formalized Mathematics*, 6(3):417–425, 1997.
25. T. Nipkow, G. Bauer, and P. Schultz. Flyspeck I: tame graphs. In *Proc. IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 2006.
26. L. Noschinski. A graph library for Isabelle. *Mathematics in Computer Science*, 9(1):23–39, 2015.
27. L. C. Paulson. A formalisation of finite automata using hereditarily finite sets. In A. P. Felty and A. Middeldorp, editors, *Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2015.
28. D. Pous and V. Vignudelli. Allegories: decidability and graph homomorphisms. In *Proc. LiCS*, pages 829–838. ACM, 2018.
29. N. Robertson and P. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325 – 357, 2004.
30. M. Sozeau. A new look at generalized rewriting in type theory. *J. Form. Reason.*, 2(1):41–62, 2009.
31. The Mathematical Components team. Mathematical components, 2017.
32. T. Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.