A Formal Proof of the Minor-Exclusion Property for Treewidth-Two Graphs

Christian Doczkal(✉), Guillaume Combette, and Damien Pous

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
{christian.doczkal,guillaume.combette,damien.pous}@ens-lyon.fr

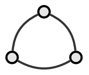
Abstract. We give a formal and constructive proof in Coq/Ssreflect of the known result that the graphs of treewidth two are exactly those that do not admit K4 as a minor. This result is a milestone towards a formal proof of the recent result that isomorphism of treewidth-two graphs can be finitely axiomatized. The proof is based on a function extracting terms from K4-free graphs in such a way that the interpretation of an extracted term yields a treewidth-two graph isomorphic to the original graph.

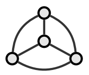
Keywords: Graph theory · Graph minor theorem · Coq · Ssreflect

1 Introduction

The notion of *treewidth* [6] 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 [1,10,13], so does model-checking of Monadic Second Order (MSO) formulae, and satisfiability of MSO formulae becomes decidable, even linear [4,5].

Robertson and Seymour's graph minor theorem [18], 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 be characterized by finite sets of excluded minors. Two standard instances are the following ones: the graphs of treewidth at most one (the forests) are precisely those excluding the cycle with three vertices (C_3); those of treewidth at most two are those excluding the complete graph with four vertices (K_4) [8].


(C_3)


(K_4)

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).

J. Avigad and A. Mahboubi (Eds.): ITP 2018, LNCS 10895, pp. 178–195, 2018.
https://doi.org/10.1007/978-3-319-94821-8_11

We present a constructive and formal proof of the latter result in Coq/Ssreflect.

Amongst the open problems related to treewidth, there is the question of finding finite axiomatisations of isomorphism for graphs of a given treewidth [5, p. 118]. This question was recently answered positively for treewidth two [14]:

$$\mathsf{K}_4\text{-free graphs form the free } 2p\text{-algebra,} \qquad (\dagger)$$

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 and contains the specific form of the graph minor theorem for treewidth two which we present here. Further, invalid proofs of related claims have already been published in the literature (see [14]). Our long term goal is to formalize (†): not only will this give us assurance about the validity of the proof in [14], it will also allow for the development of automation tactics for certain algebraic theories (e.g., 2p-algebra, allegories [11]). The Coq development accompanying the present paper [7] is a milestone for this project.

Independently from the aforementioned specific objective, formalizing the graph minor theorem for treewidth two requires us to develop a general Coq library for graph theory which should also be useful in other contexts. This library currently includes basic notions like paths, trees, subgraphs, and isomorphisms and also a few more advanced ones: minors, tree decompositions, and checkpoints (a variant of cut vertices).

We had to design this library from scratch. Indeed, there are very few formalizations of graph theory results in Coq, and none of them were applicable. Gonthier's formal proof of the Four-Color Theorem [12] is certainly the most advanced, but it restricts (by design) to planar graphs so that it cannot be used as a starting point for graph theory in the large. Similarly, Durfourd and Bertot's study of Delaunay triangulation [9] employs a notion of graphs based on hypermaps embedded in a plane. There are more formalizations in other interactive theorem provers. For instance, planar graphs were formalized in Isabelle/HOL for the Flyspeck project [16]. Noschinski recently developed a library for both simple and multigraphs in Isabelle/HOL [17]. Chou developed a large part of undirected graph theory in HOL [2]. Euler's theorem was formalized in Mizar [15]. To the best of our knowledge, the theory of minors and tree decompositions was never formalized.

Overview of the proof. We focus on connected graphs: the general case follows by decomposing any given graph into connected components. The overall strategy of our proof of the minor exclusion theorem for treewidth two is depicted in Fig. 1.

$$\begin{tikzcd}
& \text{Term} \arrow[dl, "\text{(ii)}"'] & \\
\mathsf{TW}_2 \arrow[rr, "\text{(i)}"'] & & \mathsf{K}_4\text{-free} \arrow[ul, "\text{(iii)}"']
\end{tikzcd}$$

Fig. 1. Structure of the proof.

We first prove that treewidth two graphs exclude K_4 as a minor (i). This proof is standard and relatively easy. For proving the converse implication, we introduce a notion of *term* that allow us to denote graphs. We prove that graphs of terms have treewidth at most two (ii) using properties of tree decompositions and a simple induction on terms. The main difficulty then consists in proving that every K_4-free graph can be represented by a term (iii).

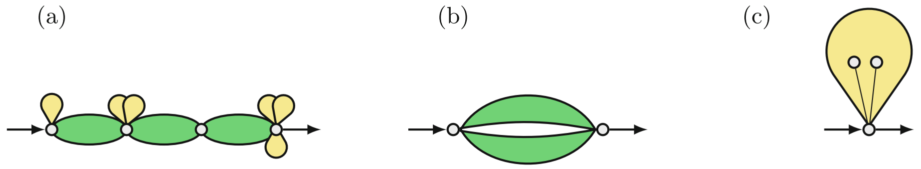
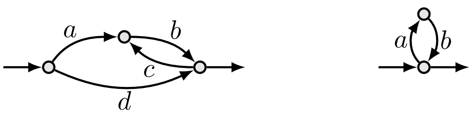


Fig. 2. The three main cases for extracting a term from a K_4-free graph.

Due to our long-term objective (†), the syntax we use for those terms is that of 2p-algebras [14];

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

This syntax makes it possible to denote directed multi-graphs, with edges labeled by letters from an alphabet Σ and with two designated vertices (the *input* and the *output*). The binary operations in the syntax correspond to series and parallel composition. 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 just two vertices. Letters represent single edges. 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 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 Fig. 2(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 green and yellow components are K_4-free whenever the starting graph is so.

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, which make it possible to proceed recursively. This is case (b) in Fig. 2. Establishing this property requires a deep analysis of the structure of K_4-free graphs.

The last case to consider (c) is when the input and the output of the graph coincide. 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.

Outline. We first discuss our representation of simple graphs and the associated library about paths; there we make use of the support for finite types from the

Ssreflect library [20], and we rely on dependent types to provide a user-friendly interface (Sect. 2). Then we proceed with our formalization of tree decompositions, minors, and associated results. This leads to implication (i) in Fig. 1, as a special instance of the fact that treewidth at most i graphs are K_{i+2}-free (Sect. 3)

Once this basic infrastructure has been set up, we move to the formalization of the concepts and results that are specific to our objective. This includes terms as well as directed labeled and possibly pointed multigraphs. We prove the implication (ii) there: terms denote graphs of treewidth at most two (Sect. 4).

As explained above, the remaining implication (iii) is the most delicate. We first establish preliminary lemmas about checkpoints and the structure of $\mathsf{K_4}$-free graphs (Sect. 5), which are then used to define an extraction function from graphs to terms (Sect. 6). Proving that this function is appropriate amounts to exhibiting a number of isomorphisms (Sect. 7).

We conclude with general remarks and statistics about the development (Sect. 8).

2 Simple Graphs

In this section we briefly describe how we represent finite simple graphs in Coq. The representation is based on finite types as defined in the Mathematical Component Libraries [20]. 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 $\mathsf{inr}\,y$) and $X_\perp$ for the *option type* (with elements $\mathsf{Some}\,x$ and None). As usual, we write $g \circ f$ for the composition of f and g. If $f : X \to Y_\perp$ and $g : Y \to Z_\perp$, we also write $g \circ f$ for the result of the monadic bind operation (with type $X \to Z_\perp$). 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. Finite types are closed under many type constructors (e.g., sum types and option types). If X is a finite type, we write 2^X for the (finite) type of sets (with decidable membership) over X. If $A : 2^X$ is a set, 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. Finite sets come with an operation $\mathsf{pick} : 2^X \to X_\perp$ yielding elements of nonempty sets and None for empty sets. Moreover, if X is a finite type and $\approx : X \to X \to \mathbb{B}$ is a boolean equivalence relation, the *quotient* [3] 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 simple graphs.

Definition 1. *A* (finite) simple graph *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 symmetric and irreflexive edge relation.*

In Coq, we represent finite graphs using dependently typed records where the last two fields are propositions:

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

We introduce a coercion from graphs to the underlying type of vertices allowing us to write $x : G$ to denote that x is a vertex of G. 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.

For sets $U : 2^G$ of vertices of G, we write $G|_U$ for the subgraph of G induced by U. This is formalized by taking the type $\Sigma x : G.\ x \in U$ of (dependent) pairs of vertices $x : G$ and proofs of $x \in U$ and lifting the edge relation accordingly. Note that 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 $x \in U$.

Definition 2. *Let G be a simple graph. 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 (i.e., the path contains no cycles). A set of vertices U is* connected *if there exists a path in U between any two vertices of U.*

The Mathematical Component Libraries include a predicate and a function

```
path : (∀ T : Type, rel T ⟶ T ⟶ seq T ⟶ bool)     last : ∀ T, T ⟶ list T ⟶ T
```

such that path $e\,x\,q$ holds if the list $x :: q$ represents a path in the relation e, and last $x\,q$ returns the last element of $x :: q$. Note that 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. This asymmetric treatment makes symmetry reasoning (using path reversal) rather cumbersome. We therefore package the path predicate and a check for the last vertex into an indexed family of types Path $x\,y$ whose elements represent xy-paths. Doing so abstracts from the asymmetry in the definition of path, makes it possible to write more compact (and thus readable) statements, helps us keeping the local context of proofs shorter, and facilitates without loss of generality reasoning.

On these packaged paths we provide (dependently typed) concatenation and reversal operations as well as an indexing operation yielding the position of the first occurrence of a vertex on the path. We define a number of splitting lemmas for packaged paths as exemplified by the lemma below.

Lemma 3. *Let p be an irredundant xy-path such that z_1 occurs before z_2 on p. Then there exists a z_2-avoiding xz_1-path, a z_1z_2-path and a z_1-avoiding z_2y-path) such that $p = p_1p_2p_3$.*

While the lemma above may seem overly specific, it is used in five different proofs (usually following some without loss of generality reasoning to order z_1 and z_2).

3 Treewidth and Minors

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

Definition 4. *A* forest *is a simple graph where there is at most one irredundant path between any two nodes.*

Definition 5. *A* tree decomposition of a simple graph G *is a forest* T *together with a function* $B : T \to 2^G$ *such that:*

T1. for every vertex $x : G$, there exists some $t : T$, such that $x \in B(t)$.
T2. for every x, the set of nodes $t : T$ such that $x \in B(t)$ is connected in T.
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.*

Note that we define the notion of tree decomposition using forests rather than trees. 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 G' can be obtained as the disjoint union of tree decompositions for G and G'.

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 partial functions inspired by [6].

Definition 6. *Let G and G' be simple graphs. A function $\phi : G \to G'_\perp$ is called a* minor map *if:*

M1. For every $y : G'$, there exists some $x : G$ such that $\phi\, x = \mathsf{Some}\, y$.
M2. For every $y : G'$, $\phi^{-1}(\mathsf{Some}\, y)$ is connected in G.
M3. If $x{-}y$ for $x, y : G'$, there exist $x_0 \in \phi^{-1}(\mathsf{Some}\, x)$ and $y_0 \in \phi^{-1}(\mathsf{Some}\, y)$ such that $x_0{-}y_0$.

G' is a minor *of G, written $G' \prec G$ if there exists a minor map $\phi : G \to G'_\perp$.*

Intuitively, the (nonempty) preimage $\phi^{-1}(\mathsf{Some}\, x)$ of a given vertex x is the (connected) set of vertices being contracted to x and the vertices mapped to None are the vertices that are removed. We sometimes use (total) minor maps $\phi : G \to G'$ corresponding to minor maps that do not delete nodes, allowing us to avoid option types in certain cases.

Making the notion of minor map explicit is convenient in that it allows us to easily construct minor maps for a given graph, starting from minor maps (with extra properties) for some of its subgraphs (cf. Lemma 29 and Proposition 30).

Definition 7. *We write* K_4 *for the complete graph with 4 vertices. A simple graph* G *is* K_4-free *if* K_4 *is not a minor of* G.

Our main result is a formal proof that a simple graph is K_4-free iff if it has treewidth at most two. We first sketch the proof that graphs of treewidth at most two are always K_4-free.

Lemma 8. *If* $\phi : G \to H_\perp$ *and* $\psi : H \to I_\perp$ *are minor maps, then* $\psi \circ \phi$ *is a minor map.*

As a consequence of the lemma above, we obtain that $\prec$ is transitive.

Lemma 9. *If* $H \prec G$*, then the treewidth of* H *is at most the treewidth of* G.

Lemma 10. *Let* T *be a forest and let* $B : T \to G$ *be a tree decomposition of* G*. Then every clique of* G *is contained in* $B(t)$ *for some* $t : T$.

The proof of Lemma 10 proceeds by induction on the size of (nonempty) cliques. For cliques of size larger than two, the proof boils down to an analysis of the set of nodes in the tree decomposition containing all vertices of the clique but one (which is nonempty by induction hypothesis) and then arguing that (due to condition T2) the removed vertex must also be present. As a consequence of Lemma 10, we have:

Proposition 11. *If* G *has treewidth at most two, then* G *is* K_4*-free.*

This corresponds to the arrow (i) in the overall proof structure (Fig. 1).

4 Graphs

In this section we define labeled directed graphs following [6]. Then we show how to interpret terms as such graphs and prove that the graphs of terms have treewidth at most two. We fix some countably infinite type of *symbols* Σ.

Definition 12. *A* graph *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 graph, we write* $x : G$ *to denote that* x *is a vertex of* G*. A* two-pointed graph *(or* 2p-graph *for short) is a structure* $\langle G, \iota, o\rangle$ *where* $\iota : G$ *and* $o : G$ *are two vertices called* input *and* output *respectively.*

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

Recall the syntax of *terms* from the introduction:

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

For each term constructor we define an operation on 2p-graphs. Those operations are depicted informally on the right of Fig. 3. For instance, $G \parallel H$, the parallel composition of G and H, consists of (disjoint) copies of G and H with the respective inputs and outputs identified. Formally, we express these graph operations in terms disjoint unions and quotients of graphs.

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

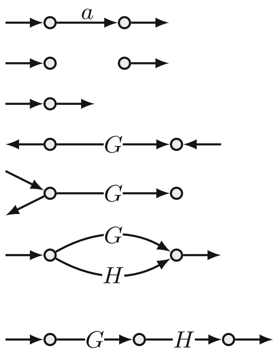


Fig. 3. The algebra of 2p-graphs.

Definition 13. *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, $s + s'$ is the pointwise lifting of s and s' to the sum type $E + E'$.

Definition 14. *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*

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

The precise definitions of the graph operations are given on the left side of Fig. 3 (A^{eqv} denotes the equivalence relation generated by the pairs in A). This allows us to interpret every term t as a 2p-graph $\mathsf{g}(t)$, recursively. We now have to prove that every 2p-graph of a term has treewidth at most two. In order to use the definition of treewidth, we first need to abstract 2p-graphs into simple graphs. This is achieved through the notion of a skeleton.

Definition 15. *Let $G = \langle V, E, s, t, l\rangle$. The* (weak) skeleton *of G 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. The weak skeleton of the 2p-graph $\langle G, \iota, o\rangle$ is the skeleton of G. The* strong skeleton *of a 2p-graph $\langle G, \iota, o\rangle$ is the skeleton of G with an additional ιo-edge.*

We remark that the operation of taking the weak or strong skeleton does not change the type of vertices. This greatly simplifies lifting properties of the skeleton to the graph and vice versa. In practice, we turn the construction of taking the weak skeleton into a coercion from graphs to simple graphs (leaving extractions of strong skeletons explicit).

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

Lemma 16. *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. □

Proposition 17. *For all terms 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 $\parallel$ and $\cdot$ follow with Lemma 16. All other cases are trivial. □

This finishes arrow (ii) of the overall proof structure (Fig. 1). The rest of the paper is concerned with arrow (iii), i.e., extracting for every 2p-graph G whose skeleton is K_4-free a term whose graph is isomorphic to G.

5 Checkpoints

Before we can define the function extracting terms from graphs, we need a number of results on simple graphs. These will allow us to analyze the structure of graphs (via their skeletons), facilitating the termination and correctness arguments for the extraction function.

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

Definition 18. *The* checkpoints *between two vertices x, y are the vertices which any xy-path must visit:*

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

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.*

Consider the graph on the left in Fig. 4; its link graph is obtained by adding the three dotted edges to the existing ones.

Note that every proper checkpoint z between vertices x and y (i.e., a vertex $z \in \mathsf{cp}\,x\,y \setminus \{x, y\}$) is a cut vertex (i.e., removing z disconnects G) and vice versa. Also note that membership in cp is decidable (i.e., $\mathsf{cp}\,x\,y$ can be defined as a finite set in the Ssreflect sense) since it suffices to check whether the finitely many irredundant paths cross z.

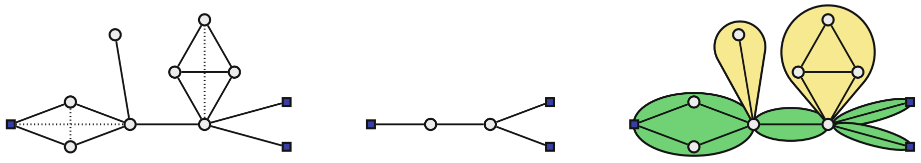


Fig. 4. Link graph, checkpoint graph, and decomposition into intervals and bags.

Lemma 19. *1.* $\mathsf{cp}\,x\,x = \{x\}$
2. $\{x, y\} \subseteq \mathsf{cp}\,x\,y = \mathsf{cp}\,y\,x$

Lemma 20. *Every irredundant cycle in the link graph is a clique.*

For a set of vertices $U \subseteq G$, we take $G+U$ to be the graph G with one additional vertex, denoted $\bullet$, whose neighbors are exactly the elements of U.

Lemma 21. *If $\{x, y, z\}$ is a triangle in the link graph, then $\mathsf{K}_4 \prec G + \{x, y, z\}$.*

Lemma 21 is first in a series of nontrivial lemmas required to justify the splitting of graphs into parallel components. Its proof boils down to an elaborate construction on paths between x, y, and z that yields a minor map from G to C_3 (the cycle with three vertices), which is subsequently extended to a minor map from $G + \{x, y, z\}$ to K_4. This is one instance where our definition of minors using minor maps pays off.

Definition 22. *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 Fig. 4 is the checkpoint graph of the one of the left, when U consists of the blue square vertices.

Lemma 23. *Let $x, y \in \mathsf{CP}\,U$. Then $\mathsf{cp}\,x\,y \subseteq \mathsf{CP}\,U$.*

We give the proof of this lemma below. It is relatively simple, but indicative of the type of reasoning required to prove checkpoint properties. Those proofs usually contain a combination of the following: splitting paths at vertices, concatenating paths, and without loss of generality reasoning. For the latter, Ssreflects `wlog`-tactic proved extremely helpful.

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

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

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 25. *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.*

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

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 Fig. 4. We first show that distinct bags and adjacent bags and strict intervals are disjoint.

Lemma 26. *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.

Lemma 27. *Let $x, y \in \mathsf{CP}\, U$. Then there exist $x_0 \in U$ and $y_0 \in U$ such that $\{x, y\} \subseteq \mathsf{cp}\, x_0\, y_0$.*

Lemma 28. *Let $\{x, y, z\}$ be a triangle in $\mathsf{CP}\, U$. Then there exist $x_0, y_0, z_0 \in U$ such that $x_0 \in [\![x]\!]_{\{x,y,z\}}$, $y_0 \in [\![y]\!]_{\{x,y,z\}}$, and $z_0 \in [\![z]\!]_{\{x,y,z\}}$.*

Proof. Follows with Lemma 27. □

Lemma 29. *Let U be nonempty and let $T \triangleq U \cup (G \setminus \bigcup_{x \in U} [\![x]\!]_U)$. Then there exists a minor map $\phi : G \to G|_T$ such that ϕ maps the elements of each bag $[\![x]\!]_U$ to x and every other vertex to itself.*

The above series of lemmas leads us to the following proposition, that corresponds to [14, Proposition 20(i)]; the proof given here is significantly simpler than the proof given in [14].

Proposition 30. *Let $U \subseteq G$ such that $G + U$ is K_4-free. Then $\mathsf{CP}\,U$ is a tree.*

Proof. Assume that $\mathsf{CP}\,U$ is not a tree. Then $\mathsf{CP}\,U$ contains a triangle $\{x, y, z\}$ (Lemma 20). Let x_0, y_0, z_0 as given by Lemma 28. We obtain a minor map collapsing the bags for x, y, and z (Lemma 29 with $U = \{x, y, z\}$). This identifies x and x_0 and likewise for y and z. Since x, y, z is still a triangle in the link graph of the collapsed graph and since $\bullet$ is adjacent to x, y, z in the collapsed graph, Lemma 21 yields $\mathsf{K}_4 \prec G + U$, a contradiction. $\square$

The following proposition establishes the key property of K_4-free graphs we alluded to in the introduction. Its proof is particularly tricky to formalize due to the number of different graphs with shared vertices (we have G, $G' \triangleq G|_{\overline{\{i\}}}$ and $G' + U$ (the graph Proposition 30 is instantiated with). Consequently, we often need to cast vertices from one graph to another.

Proposition 31. *Let $\iota, o : G$ such that $G + \iota o$ is K_4-free, $\llbracket \iota \rrbracket_{\{\iota, o\}} = \{\iota\}$, and $\iota \lozenge o$, but not $\iota \text{—} o$. Then $\rrbracket \iota; o \llbracket$ has at least two connected components.*

Proof. Let G' be the graph G with ι removed and let $U \subseteq G'$ be the set of neighbors of ι (in G) plus o. By Proposition 30 (on G' and U), $\mathsf{CP}\,U$ is a tree in G'. The vertex o cannot be a leaf in $\mathsf{CP}\,U$ since if it were, its unique neighbor would be a proper checkpoint between ι and o. Moreover, o is a checkpoint between any distinct neighbors of o. Removing o yields that $\rrbracket \iota; o \llbracket$ has at least two components. $\square$

The above proposition is used for splitting paths into parallel components (case (b) in Fig. 2); the one below allows us to proceed recursively in case (a).

Proposition 32. *Let $\iota, o : G$ such that $G + \iota o$ is K_4-free and let $x, y \in \mathsf{cp}\,\iota\,o$ such $x \neq y$. Then $\llbracket x; y \rrbracket + xy$ is K_4-free.*

Proof. Without loss of generality x appears before y on every ιo-path. We obtain that $\llbracket x; y \rrbracket + xy$ is a minor of $G + \iota o$ by collapsing $\llbracket x \rrbracket_{\{x,y\}}$ (which contains ι) to x and $\llbracket y \rrbracket_{\{x,y\}}$ (which contains o) to y (Lemma 29). $\square$

6 Extracting Terms from K_4-free Graphs

We say that a 2p-graph G is *CK4F* if its skeleton is connected and its strong skeleton is 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 aT ⟶ 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).[1] 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 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 Fig. 2. 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. We use the measure below to justify termination.

Definition 33. *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 is then defined as follows:

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

where the definition of F is given in Fig. 5. 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 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

[1] 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.

```
Definition $\mathsf{F}(t : \text{2p-graph} \to \text{term})(G : \text{2p-graph}) \triangleq$
    let $\langle\langle V, E', s, t, l\rangle, \iota, o\rangle := G$ in
    if $\iota = o$ then
        let $E := \mathcal{E}(\{\iota\})$ in
        if $E = \emptyset$ then
            if $\mathsf{pick}(\mathsf{components}(V \setminus \{i\}))$ is $\mathsf{Some}\,C$ then
                $\mathrm{dom}(t(\mathsf{redirect}\,C)) \parallel t(G[\overline{C}])$
            else 1
        else (* $E \neq \emptyset$ *)
            $\left(\|_{e \in E}\, \mathsf{tm}(e)\right) \parallel G[V, \overline{E}]$
    else (* $i \neq o$ *)
        if $\mathcal{E}(\llbracket\iota\rrbracket_{\{\iota,o\}}) = \emptyset \wedge \mathcal{E}(\llbracket o\rrbracket_{\{\iota,o\}}) = \emptyset \wedge \mathsf{cp}\,\iota\,o = \{\iota, o\}$ then
            let $P := \mathsf{components}(\,\rrbracket\iota; o\llbracket\,)$ in
            let $E := \mathcal{E}(\{\iota, o\})$ in
            if $E = \emptyset$ then
                if $\mathsf{pick}\,P$ is $\mathsf{Some}\,C$ then
                    $t(\mathsf{component}(C)) \parallel t(G[\overline{C}])$
                else 1 (* never reached *)
            else (* $E \neq \emptyset$ *)
                if $P = \emptyset$ then
                    $\|_{e \in E}\, \mathsf{tm}(e)$
                else
                    $\left(\|_{e \in E}\, \mathsf{tm}(e)\right) \parallel t(G[V, \overline{E}])$
        else (* nontrivial $\iota$ or $o$-bag or proper checkpoint between $\iota$ and $o$ *)
            if $\mathcal{E}(\llbracket\iota\rrbracket_{\{\iota,o\}}) \neq \emptyset \vee \mathcal{E}(\llbracket o\rrbracket_{\{\iota,o\}}) \neq \emptyset$ then
                $t(G[\iota]) \cdot t(G[\iota, o]) \cdot t(G[o])$
            else
                if $\mathsf{pick}\,(\mathsf{cp}\,\iota\,o \setminus \{\iota, o\})$ is $\mathsf{Some}\,z$ then
                    $t(G[\iota, z]) \cdot t(G[z]) \cdot t(G[z, o])$
                else 1 (* never reached *)
```

Fig. 5. The term extraction function

of G (not the strong skeleton). For a given 2p-graph $G = \langle G', \iota, o\rangle$, we also define:

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

Note that $\mathsf{component}(C)$ is obtained as 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 34. *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), Proposition 31 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), Proposition 32 ensures that the strong skeletons of $G[\iota, z]$ and $G[z, o]$ are K_4-free.

As a consequence of Lemma 34, we obtain:

Proposition 35. *Let G be CK4F. Then $\mathsf{t}\,G = \mathsf{F}\,\mathsf{t}\,G$.*

7 Isomorphism Properties

In this section we establish that interpreting the terms extracted from a 2p-graph G yields a graph that is isomorphic to G. This is the part of the proof 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.

Definition 36. *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 $l \equiv l' \circ g$. A* homomorphism *from $\langle G, \iota, o\rangle$ to $\langle G', \iota', o'\rangle$ is a graph homomorphism $\langle f, g\rangle$ from G to G' respecting inputs and outputs: $f(\iota) = \iota'$ and $f(o) = o'$.*

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 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 $\|$ 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 isomorphism properties.

We first establish that all graph operations respect isomorphism classes.

Lemma 37. *Let* $G_1 \simeq G'_1$ *and* $G_2 \simeq G'_2$*. Then we have* $G_1 \parallel G_2 \simeq G'_1 \parallel G'_2$, $G_1{\cdot}G_2 \simeq G'_1{\cdot}G'_2$*, and* $\mathrm{dom}(G_1) \simeq \mathrm{dom}(G'_1)$*.*

Lemma 37 allows rewriting with isomorphisms underneath the graph operations using Coq's generalized (setoid) rewriting tactic [19].

The proofs for establishing that two graphs (of some known shape) are isomorphic generally follow the same pattern: define the pair of functions $\langle f, g\rangle$ (cf. Definition 36) as well as their respective inverses and then show all the required equalities (including that the proposed inverses are indeed inverses). This amounts to 9 equalities per isomorphism that all need to be verified. Additional complexity is introduced by the fact that we are almost exclusively interested in isomorphism properties involving $\parallel$ and $\cdot$ which are defined using quotient constructions. Among others, we establish the following isomorphism lemmas:

Lemma 38. *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 39. *Let* $G = \langle G', \iota, o\rangle$ *such that* $\mathcal{E}([\![\iota]\!]_{\{\iota,o\}}) = \emptyset$, $\mathcal{E}([\![o]\!]_{\{\iota,o\}}) = \emptyset$*, 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 40. *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 41. *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 42. *Let* G *be a 2p-graph. Then* $\mathsf{g}(\mathsf{t}\,G) \simeq G$*.*

Proof. By induction on the measure of G. We use Proposition 35 to unfold the definition of t. Each of the cases follows with the induction hypothesis (using the lemmas underlying the proof of Lemma 34 to justify that the induction hypothesis applies) and some isomorphism lemmas (e.g., Lemmas 37 to 41). □

Note that Lemma 40 justifies both the split in line 7 and the split in line 17 (in the latter case $]\!]\iota; o[\![= \overline{\{\iota, o\}}$).

Putting everything together, we obtain our main result.

Theorem 43. *A simple graph is* K_4*-free iff if it has treewidth at most two.*

Proof. Fix some simple graph G. The direction from right to left follows with Proposition 11. For the converse direction we proceed by induction on $|G|$. If G is connected (and nonempty; otherwise the claim is trivial), we construct a 2p-graph (with $\iota = o$) whose (strong) skeleton is isomorphic to G. By Theorem 42, the skeleton of $\mathsf{g}(\mathsf{t}\,G)$ is isomorphic to G and, hence, K_4-free by Proposition 17. If G contains disconnected vertices x and y, then G is isomorphic to the disjoint union of the connected component containing x and the rest of the graph (which must contain y). The claim then follows by induction hypothesis using the fact that treewidth is preserved under disjoint union. □

Note that Theorem 42 is significantly stronger than what is needed to establish Theorem 43. To prove the latter, it would be sufficient to extract terms that can be interpreted as simple graphs, thus avoiding the complexity introduced by labels, edge multiplicities and loops. The fine-grained analysis we formalize here is however crucial for our long-term objective (†).

8 Conclusion

We have developed a library for graph theory based on finite types as provided by the Mathematical Components Libraries. As a major step towards proving that K_4-free 2p-graphs form the free $2p$-algebra (†), we gave a proof of the graph-minor theorem for treewidth two, using a function extracting terms from K_4-free graphs.

The Coq development accompanying this paper [7] consists of about 6700 lines of code, with a ratio of roughly 1:2 between specifications and proofs. It contains about 200 definitions and about 550 lemmas. Many of these have short proofs, but some proofs (e.g., the proof of Proposition 31) are long intricate constructions without any obvious lemmas to factor out. As mentioned before, the isomorphism proofs for Sect. 7 mostly follows the same pattern. Hence, we hope that they can be automated to some degree.

As it comes to proving (†), there are two main challenges to be solved. First we should prove that the choices made by the extraction function are irrelevant modulo the axioms of 2p-algebras (e.g., which neighbor is chosen in $\mathsf{redirect}(C)$). This is why we were careful to define this function as deterministically as possible. Second, we should prove that it is a homomorphism (again, modulo the axioms of 2p-algebras). Those two steps seem challenging: their paper proofs require a lot of reasoning modulo graph isomorphism [14].

References

1. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. Theoret. Comput. Sci. **239**(2), 211–229 (2000). https://doi.org/10.1016/S0304-3975(99)00220-0
2. Chou, C.-T.: A formal theory of undirected graphs in higher-order logc. In: Melham, T.F., Camilleri, J. (eds.) HUG 1994. LNCS, vol. 859, pp. 144–157. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58450-1_40
3. Cohen, C.: Pragmatic quotient types in Coq. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 213–228. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39634-2_17
4. Courcelle, B.: The monadic second-order logic of graphs. I: recognizable sets of finite graphs. Inf. Comput. **85**(1), 12–75 (1990). https://doi.org/10.1016/0890-5401(90)90043-H
5. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach. Encyclopedia of Mathematics and Its Applications, vol. 138. Cambridge University Press, Cambridge (2012)

6. Diestel, R.: Graph Theory, Graduate Texts in Mathematics. Springer, New York (2005)
7. Doczkal, C., Combette, G., Pous, D.: Coq formalization accompanying this paper. https://perso.ens-lyon.fr/damien.pous/covece/k4tw2
8. Duffin, R.: Topology of series-parallel networks. J. Math. Anal. Appl. **10**(2), 303–318 (1965). https://doi.org/10.1016/0022-247X(65)90125-3
9. Dufourd, J.-F., Bertot, Y.: Formal study of plane delaunay triangulation. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 211–226. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_16
10. Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: NCAI, pp. 4–9. AAAI Press/The MIT Press (1990)
11. Freyd, P., Scedrov, A.: Categories, Allegories. North Holland, Elsevier, Amsterdam (1990)
12. Gonthier, G.: Formal proof – the four-color theorem. Notices Amer. Math. Soc. **55**(11), 1382–1393 (2008)
13. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM **54**(1), 1:1–1:24 (2007). https://doi.org/10.1145/1206035.1206036
14. Llópez, E.C., Pous, D.: K4-free graphs as a free algebra. In: MFCS. LIPIcs, vol. 83, pp. 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.MFCS.2017.76
15. Nakamura, Y., Rudnicki, P.: Euler circuits and paths. Formalized Math. **6**(3), 417–425 (1997)
16. Nipkow, T., Bauer, G., Schultz, P.: Flyspeck I: tame graphs. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 21–35. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_4
17. Noschinski, L.: A graph library for Isabelle. Math. Comput. Sci. **9**(1), 23–39 (2015). https://doi.org/10.1007/s11786-014-0183-z
18. Robertson, N., Seymour, P.: Graph minors. XX. Wagner's conjecture. J. Comb. Theor. Ser. B **92**(2), 325–357 (2004). https://doi.org/10.1016/j.jctb.2004.08.001
19. Sozeau, M.: A new look at generalized rewriting in type theory. J. Form. Reason. **2**(1), 41–62 (2009). https://doi.org/10.6092/issn.1972-5787/1574
20. The Mathematical Components Team: Mathematical Components (2017). http://math-comp.github.io/math-comp/