

Completeness of an Axiomatization of Graph Isomorphism via Graph Rewriting in Coq

Christian Doczkal, Damien Pous

▶ **To cite this version:**

Christian Doczkal, Damien Pous. Completeness of an Axiomatization of Graph Isomorphism via Graph Rewriting in Coq. 2019. hal-02333553v1

HAL Id: hal-02333553

https://hal.science/hal-02333553v1

Preprint submitted on 25 Oct 2019 (v1), last revised 8 Jan 2020 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Completeness of an Axiomatization of Graph Isomorphism via Graph Rewriting in Coq

Damien Pous
Plume team, CNRS, LIP, ENS de Lyon

Christian Doczkal
Université Côte d'Azur, Inria Sophia Antipolis - Marelle

Abstract

The labeled multigraphs of treewidth at most two can be described using a simple term language over which isomorphism of the denoted graphs can be finitely axiomatized. We formally verify soundness and completeness of such an axiomatization using Coq and the mathematical components library. The completeness proof is based on a normalizing and confluent rewrite system on term-labeled graphs. While for most of the development a dependently typed representation of graphs based on finite types of vertices and edges is most convenient, we switch to a graph representation employing a fixed type of vertices shared among all graphs for establishing confluence of the rewrite system. The completeness result is then obtained by transferring confluence from the fixed-type setting to the dependently typed setting.

The library is available from

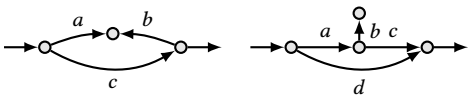
http://perso.ens-lyon.fr/damien.pous/covece/graphs/

1 Introduction

In graph theory, the notion of *treewidth* [6] of a graph measures how close a graph is to a forest. In particular, the graphs of treewidth at most one are just the forests. Among the open problems related to treewidth, there is the question of finding finite axiomatisations of isomorphism for graphs of a given treewidth [5, page 118]. This question was recently answered positively for treewidth two [4], in the following way. A 2pdom-algebra is an algebra over the following signature, subject to eleven equational axioms:

$$\Sigma = \{\cdot_2,\ \|_2, (_^\circ)_1, \mathrm{dom}_1, 1_0\}$$

Graphs can be shown to form an algebra for this signature, so that a term u over this signature makes it possible to denote a graph $\mathrm{g}(u)$. For instance, the graphs of $(a{\cdot}b^\circ) \parallel c$ (left) and $(a{\cdot}\mathrm{dom}(b){\cdot}c) \parallel d$ (right) are:



A first important result is that Σ-terms make it possible to denote precisely the class of connected graphs of treewidth at most two. That the graph of a term is connected and has treewidth at most two is relatively easy; the converse direction—every connected graph of treewidth at most two can be represented by a term—is much harder [4]. This result was recently formalized in Coq [7], by going through the well-known characterization of treewidth at most two graphs as those excluding K_4 as a minor [10].

We also have that graphs modulo isomorphism form a 2pdom-algebra: the eleven axioms of 2pdom-algebra are all valid in the algebra of graphs. While a pen and paper proof of this result is relatively easy, formalizing it in a proof assistant is non-trivial: it requires proper tools for combining graphs (to get the algebra of graphs) and for establishing graph isomorphisms between graphs obtained by nested quotients and disjoint unions (to get the laws). This was also formalized in Coq, in [9].

As a consequence, for all terms u, v which are provably equal from the 2pdom-axioms, $\mathrm{g}(u)$ and $\mathrm{g}(v)$ are isomorphic. The converse implication also holds: 2pdom-axioms are complete w.r.t. graph isomorphisms, so that (connected, treewidth at most two) graphs actually form the free 2pdom-algebra. Formalizing this completeness theorem in Coq is the main contribution of the present paper.

This theorem is difficult because it must translate a rather global notion (an isomorphism between two graphs) into a sequence of local reasoning steps (an equational proof from 2pdom-axioms). A first proof of this result was given in [4], which Doczkal and Pous hoped to formalize in Coq [7] until they realized an alternative proof could be used [8]. This latter proof is much easier from the graph-theoretical point of view: it does not require a precise analysis of the structure of treewidth at most two graphs. Instead, it relies on a graph rewrite system which happens to be confluent modulo 2pdom-axioms. By using Newman's lemma to prove confluence via local confluence, the starting global isomorphism is only analyzed locally in the proof. This is the proof we follow in the present formalization; we give a more precise sketch of it in Section 3.

That this proof is simpler does not mean it is easy to formalize in a proof assistant: it involves both local and global operations on graphs, and the local confluence proof, which is already long on paper, requires good abstractions in order not to get lost in the details.

In order to define the rewrite system and prove most of the required results about it (Sections 4 to 7), we need to slightly generalize the letter-labeled graphs from [7, 9] to allow for a uniform treatment of both letter-labeled graphs (used in the statement) and 2pdom-labeled graphs (used for the rewrite system). However, even the generalized representation turned out to be inappropriate for getting a formal proof of local confluence. We circumvent this difficulty by

$$x \parallel (y \parallel z) = (x \parallel y) \parallel z \qquad x \parallel y = y \parallel x \qquad 1 \parallel 1 = 1$$

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \qquad x \cdot 1 = x$$

$$x^{\circ\circ} = x \qquad (x \parallel y)^\circ = x^\circ \parallel y^\circ \qquad (x \cdot y)^\circ = y^\circ \cdot x^\circ$$

$$\mathrm{dom}(x \parallel y) = 1 \parallel x \cdot y^\circ \qquad \mathrm{dom}(x \cdot y) = \mathrm{dom}(x \cdot \mathrm{dom}(y))$$

$$\mathrm{dom}(x) \cdot (y \parallel z) = \mathrm{dom}(x) \cdot y \parallel z$$

Figure 1. Axioms of 2pdom-algebras.

using a second representation for graphs and rewrite steps for this part of the proof. This second representation makes it possible to prove local confluence in a natural way, as with pen and paper, by organizing case distinctions as most appropriate and using 'without-loss-of-generality' reasoning to factor out similar cases (Section 8). We then establish correspondence between our two representations (Section 9), allowing us to transfer local confluence from the second representation to the first and wrap everything together.

2 2pdom-algebras

We first recall the definition and basic properties of 2pdom-algebra [4, 16]. We consider the signature from the introduction for terms and algebras. We sometimes omit the $\cdot$ symbol and we assign priorities so that the expression $(x \cdot (y^\circ)) \parallel z$ can be written just as $xy^\circ \parallel z$.

Definition 2.1. A 2pdom-algebra is a Σ-algebra satisfying the axioms from Fig. 1.

Definition 2.2. An element x of a 2*pdom*-algebra is called a *test* if $x \parallel 1 = x$. We let $\alpha, \beta, \ldots$ range over such tests.

Note that 1 is a test, as well as $\mathrm{dom}(x)$, for all x.

Lemma 2.3. *For all tests α, β in a 2pdom-algebra, we have*

$$\alpha^\circ = \alpha \qquad \alpha\beta = \alpha \parallel \beta = \beta\alpha$$

We deduce from the second equation that $\alpha\beta$ is a test, and that tests with $\cdot$ and 1 form a commutative monoid.

We fix an alphabet $\mathcal{A}$ in the remainder, with letters $a, b, \ldots$. We let $u, v \ldots$ range over Σ-terms with variables in $\mathcal{A}$, which we call *terms* in the sequel. Given two such terms u, v, we write $u \equiv v$ when the equation is derivable from the axioms of 2pdom-algebras (equivalently, when the equation universally holds in all 2pdom-algebras). Terms quotiented by this relation form the free 2pdom-algebra over $\mathcal{A}$.

3 Sketch of completeness proof

We now sketch the completeness proof from [8], delegating most of the formal definitions to the subsequent sections.

We consider directed multigraphs with two designated vertices respectively called input and output, and edges labeled in $\mathcal{A}$. We just call them graphs. Examples of such

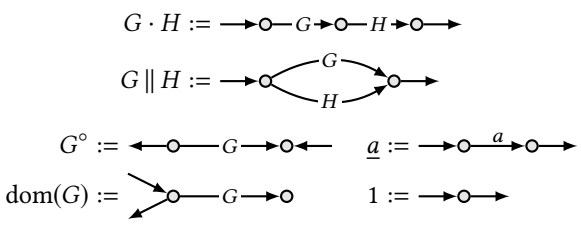


Figure 2. Graph operations.

graphs were given in Section 1, where inputs and outputs were depicted using unlabeled ingoing and outgoing arrows.

Graphs form a 2pdom-algebra by considering the operations in Fig. 2. The binary operations $(\cdot)$ and $(\parallel)$ respectively correspond to series and parallel composition, *converse* $(_^\circ)$ just exchanges input and output, and *domain* (dom) relocates the output to the input.

Proposition 3.1 (Soundness)**.** *Graphs form a 2pdom-algebra.*

By interpreting a letter $a \in \mathcal{A}$ as the graph $\underline{a}$ in Fig. 2, we can associate a graph $\mathrm{g}(u)$ to every term u (cf. rest of Fig. 2).

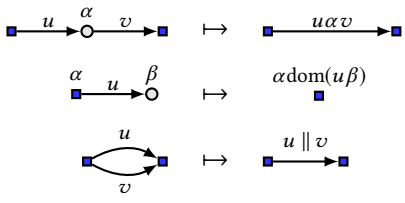
Note that the parallel composition of a graph with the graph 1 merges the input and output of the former graph. For instance, the graph $\underline{a} \parallel 1$ consists of a single vertex with a self-loop labeled with a. We have that a term u is a test if and only if the input and output of $\mathrm{g}(u)$ coincide [8, Lemma 10].

By Proposition 3.1, $\mathrm{g}(u)$ and $\mathrm{g}(v)$ are isomorphic whenever $u \equiv v$. A formal proof of this soundness result is described in [9]. We slightly generalize it in the present work (Proposition 5.2 below), but our main contribution is a formal proof that the converse implication also holds: the axioms of 2pdom-algebra are complete w.r.t. graph isomorphism:

Theorem 3.2 (Completeness)**.** *For all terms u, v such that $\mathrm{g}(u)$ and $\mathrm{g}(v)$ are isomorphic, we have $u \equiv v$.*

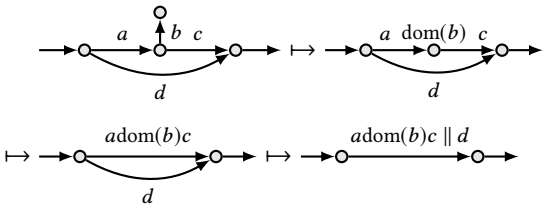
As explained in the introduction, the key idea in the proof from [8] consists in using a graph rewrite system. This system is not used to obtain canonical normal forms. Instead, it makes it possible to recover various terms denoting a given graph, and its confluence makes it possible to relate those terms via the axioms.

The rewrite system works on a generalization of the previous graphs, where vertices are labeled with tests and edges are labeled by terms rather than letters. Its main rules are the following ones:



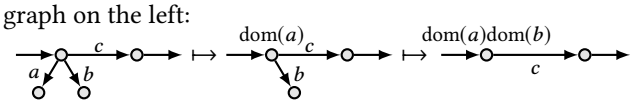
(The circular vertices, i.e., those that are removed, must be distinct from input and output and must not have other incident edges.) Its behavior is reminiscent from the state-removal algorithm used to construct a regular expression from an automaton: vertices and edges are removed until we obtain a small graph, from which we can read back a term.

For instance, we have the following sequence of rewrite steps, which witnesses the fact that $a{\cdot}\mathrm{dom}(b){\cdot}c \parallel d$ is a term denoting the starting graph.



Note that the rewrite system is non-deterministic: one of the most trivial examples is that we could also obtain the term $d \parallel a{\cdot}\mathrm{dom}(b){\cdot}c$ from the above graph. It is however confluent modulo a generalized notion of isomorphism where labels are compared using the relation $\equiv$. Proving this property is one of the key steps of the completeness proof.

Also note that there are terms which cannot be reached via the rewrite system. Consider for instance, the reduction below, which gives us the term $(\mathrm{dom}(a){\cdot}\mathrm{dom}(b)){\cdot}c$ for the graph on the left:



A similar reduction would give us the term $(\mathrm{dom}(b){\cdot}\mathrm{dom}(a)){\cdot}c$; but no reduction would give us $\mathrm{dom}(a){\cdot}(\mathrm{dom}(b){\cdot}c)$, which is also a term denoting the same graph. To deal with this issue, we use a syntactic normalization function $u \mapsto u_\downarrow$ in order to isolate the tests occurring on both sides of a term. (Normalized terms are not canonical, though: the normalization function is far from equating all provably equal terms.)

All in all, the completeness proof is obtained by combining the following three key properties:

(i) for all u, $u \equiv u_\downarrow$
(ii) for all u, $\mathrm{g}(u) \mapsto^* \mathrm{g}'(u_\downarrow)$, where for a normalized term v, $\mathrm{g}'(v)$ is the small irreducible graph labeled with v.
(iii) $\mapsto$ is confluent modulo generalized isomorphism

The first one is a syntactic property proved by induction on u, whose formalization requires some work but does not raise any problem (induction and equational reasoning have very good support in proof assistants).

The second one, *reducibility*, is also proved by induction on u, but requires much more work: we have to prove that rewrite steps are preserved under the various algebraic operations, and that there are enough rewrite rules to effectively reduce every graph of a term into a small irreducible graph.

We use Newman's lemma to reduce confluence to local confluence. Then we have to analyze all critical pairs, and show that they can all be joined modulo isomorphism and 2pdom axioms. This is the most delicate step for the formalization: we have to analyze many cases, and in each case we need to produce appropriate rewrite steps and isomorphisms.

Once we have proved those three properties, completeness is obtained as follows. Starting from two terms such that $\mathrm{g}(u)$ and $\mathrm{g}(v)$ are isomorphic, we know by reducibility (ii) that $\mathrm{g}(u) \mapsto^* \mathrm{g}'(u_\downarrow)$ and $\mathrm{g}(v) \mapsto^* \mathrm{g}'(v_\downarrow)$. Since the graphs $\mathrm{g}'(u_\downarrow)$ and $\mathrm{g}'(v_\downarrow)$ are irreducible, we deduce by confluence (iii) that they are isomorphic (modulo the axioms), from which we deduce $u_\downarrow \equiv v_\downarrow$. We conclude that $u \equiv v$ by transitivity, using (i) twice.

4 Packaged Labeled Multigraphs

We now describe the formalization of labeled multigraphs. As explained above, we need two kinds of graphs: graphs with unlabeled vertices and edges labeled in $\mathcal{A}$ for the overall completeness result, as well as graphs with tests-labeled vertices and term-labeled edges as an intermediate data-structure. In order to share code and define the various operations on those two classes of graphs at once, we introduce the following abstraction for labels, which we explain below:

Definition 4.1 (Label Structure)**.** A *setoid* is a pair $(X, \equiv)$ of a type X and an equivalence relation $\equiv$ on X. A *label structure* consists of the following

- two setoids $(L_v, \equiv_v)$ and $(L_e, \equiv_e)$;
- a binary operation $\otimes$ and an element $1 : L_v$ such that $(L_v, \equiv_v, \otimes, 1)$ forms a commutative monoid up to $\equiv_v$;
- a symmetric relation $\equiv'_e$ on L_e such that $\equiv_e \equiv'_e \subseteq \equiv'_e$ and $\equiv'_e \equiv'_e \subseteq \equiv_e$.

We usually omit the indices on equivalence relations, if they can be inferred from the context. We use setoids for labels since we want to instantiated these labels with tests and terms, compared modulo the axioms of 2pdom-algebra.

We require a commutative monoid on vertex labels because one of the most crucial operations in our development is that of forming vertex-quotients of graphs (i.e., collapsing the vertices of a graph with respect to some equivalence relation): collapsed vertices get labeled with the composition in the monoid of their initial labels.

The last requirement is motivated as follows. In order to express the rewrite system succinctly and to enable more opportunities to reason by symmetry, it is convenient to consider a notion of isomorphism on term-labeled graphs where edges can be flipped, using the converse operation to update the label: an edge from x to y labeled with u can be seen as an edge from y to x labeled with u°. In contrast, such an operation should not be allowed on letter-labeled graphs (there is no converse operation on the alphabet).

Using the relation $\equiv'_e$ makes it possible to capture those two cases. We will define isomorphisms such that a u labeled edge from x to y can be mapped to a v labeled edge from y to x provided $u \equiv'_e v$. The requirements on this relation ensure that we obtain an equivalence relation on graphs

when doing so. The two label structures below capture the two aforementioned cases.

Definition 4.2 (The alphabet $\mathcal{A}$ as a label structure)**.** The trivial monoid (on the single element type `unit`), together with the discrete setoid on $\mathcal{A}$ (i.e., with Leibniz equality), and the empty relation on $\mathcal{A}$ for $\equiv'$, form a label structure.

Definition 4.3 (2pdom-algebras as label structures)**.** For every 2pdom-algebra X, we have a label structure with X as edge-labels, tests as vertex-labels, and $x \equiv'_e y := x \equiv y^\circ$.

We assume a label structure L for the rest of this section and the following one, with setoids $(L_v, \equiv)$ and $(L_e, \equiv)$.

Definition 4.4 (Graph)**.** An *(L-labeled directed multi-) graph* is a structure $G = \langle V, E, p, l^v, l^e\rangle$, where

- V is a finite type of *vertices*
- E is a finite type of *edges*
- $p : \mathbb{B} \to E \to V$ is a function where p false e indicates the *source* of the edge e and p true e indicates the *target* of the edge e
- $l^v : V \to L_v$ indicates the *label* of each vertex
- $l^e : E \to L_e$ indicates the *label* of each edge

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. Representing the source and target functions for edges using a single function allows us to avoid code duplication at several places: this pieces of information are often handled in a uniform way.

In order to define sequential and parallel composition (cf. Fig. 2), we rely, as in [7, 9] on the following two operations: disjoint union and (vertex-)quotients.

Definition 4.5. Let $G{=}\langle V, E, p, l^v, l^e\rangle$ and $G'{=}\langle V', E', p', l^{v\prime}, l^{e\prime}\rangle$. The *disjoint union* of G and G', is the graph

$$G + G' := \langle V + V', E + E', p + p', l^v + l^{v\prime}, l^e + l^{e\prime}\rangle$$

Here, $f + f'$ (for $f \in \{p, l^v, l^e\}$) is the pointwise lifting of f and f' to the sum type $E + E'$ or $V + V'$ with results in $V + V'$, L_v, or L_e).

In order to define quotients on graphs we exploit that finite types are closed under taking quotients. If $\approx : X \to X \to \mathbb{B}$ is a boolean equivalence relation on some finite type X, the *quotient* [2] 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 $x : X$.

Definition 4.6. Let $G{=}\langle V, E, p, l^v, l^e\rangle$ and let $\approx : G \to G \to \mathbb{B}$ be an equivalence relation. The *quotient of G modulo $\approx$*, written $G_{/\approx}$, is the graph

$$\langle V_{/\approx}, E, \lambda be.\, \pi(p\, b\, e), l^{v\prime}, l^e\rangle$$

where $l^{v\prime} := \lambda x.\ \bigotimes_{(y:V\,|\,\pi y = x)} l^v\, y$ gathers in every equivalence class (i.e., every vertex of the new graph) all the vertex labels of the vertices in the class.

In addition to those global operations for the algebra of graphs, we need the following local operations to define the rewrite system:

Definition 4.7. Let G be a graph and let $x, y : G$, $\alpha, \beta : L_v$, and $u : L_e$. We write:

- 1_α for the edge-free graph with one α-labeled vertex.
- $2^\beta_\alpha := 1_\alpha + 1_\beta$ (a graph with only two vertices).
- $G \dotplus \alpha := G + 1_\alpha$ (G with an additional vertex).
- $G \dotplus [x, u, y]$ for G with an additional u-labeled edge from x to y.
- $u^\beta_\alpha := 2^\beta_\alpha \dotplus [\mathsf{in}_l *, u, \mathsf{in}_r *]$ (a graph with a single edge between two distinct vertices).
- $G[x \leftarrow \alpha]$ for G where x is labeled with $\alpha \otimes l^v x$ (i.e, α is combined with the existing label of x).

Note that for all these graph operations (both the local ones and the global ones), working in Coq with a representation of graphs closely following Definition 4.4 is extremely convenient: it is compact, and all basic invariants are nicely enforced via (dependent) types. Moreover, the MathComp [18] library provides all the required infrastructure for taking disjoint unions and quotients on finite types.

Given a boolean $b : \mathbb{B}$, we write $u \equiv_{[b]} v$ for $u \equiv v$ if is b = false, or $u \equiv' v$ otherwise. This allows us to define isomorphisms as follows:

Definition 4.8. Let $F{=}\langle V, E, p, l^v, l^e\rangle$ and $G{=}\langle V', E', p', l^{v\prime}, l^{e\prime}\rangle$ be graphs. A *homomorphism* from F to G consists of three functions $h^v : V \to V'$, $h^e : E \to E'$, and $h^d : E \to \mathbb{B}$ s.t.

1. for all $e : E$, $b : \mathbb{B}$, $p'\, b\, (h^e\, e) = h^v(p\, (h^d\, e \oplus b)\, e)$.
2. for all $x : V$, $l^{v\prime}(h^v\, x) \equiv h^v\, x$.
3. for all $e : E$, $l^{e\prime}(h^e\, e) \equiv_{[h^d\, e]} l^e\, e$.

An *isomorphism* is a homomorphism where h^v and h^e are bijections. We denote the type of isomorphisms between graphs F and G by $F \simeq G$.

Intuitively, the h^d predicate in a homomorphism indicates whether the homomorphism flips a given edge or not. If an edge e is flipped, the vertex component h^v should map the source of e to the target of $h^e\, e$ and vice versa, whence the use of a boolean xor operation in the first requirement. Accordingly, the labels of the two edges should be related by $\equiv'$ rather than $\equiv$ when an edge is flipped (third requirement).

Note that the notion of isomorphism depends on the label structure. When the label structure L is the one from Definition 4.2, h^d must be the constantly false function (because of the third condition), and the second condition vanishes since all vertices are labeled with the unique value of type `unit`; therefore, we recover in this case a standard definition of non-edge-flipping isomorphism on edge-labeled multigraphs [7, 9].

Fact 4.9. *Graph isomorphism is an equivalence relation.*

We remark that Definition 4.8 formalizes the computational notion of isomorphism rather the property of two graphs being isomorphic. This is crucial for a compositional treatment of isomorphisms. To see this, consider the graph expression $F \dotplus [x, u, y]$ and the operation of replacing F with an isomorphic graph G. This requires to also replace the vertices x and y with their respective images under the vertex component of the isomorphism. This approach is thus required in order to state (and prove) the two congruence properties below. (In the following, when an isomorphism $h : F \simeq G$ appears as a function, it is to be taken as the underlying vertex component h^{υ}.)

Lemma 4.10. *Let F, G be graphs and let $h : F \simeq G$.*

- $F \dotplus [x, u, y] \simeq G \dotplus [h\,x, u, h\,y]$
- $F[x \leftarrow \alpha] \simeq G[h\,x \leftarrow \alpha]$

Concretely in Coq, formalizing the isomorphisms in such a computational way requires us

- to place the definition in Type rather than Prop,
- to express that functions are bijective using explicit inverses (here we build on a small library where we encapsulate computational bijections between types),
- to make sure that whenever we define an isomorphism, we make it either transparent so that its computational content is immediately available, via reduction, or we prove appropriate equations about it before making it opaque for reduction.

Like in [9], we prove the following properties about the global operations of union and quotient:

Lemma 4.11 (Generalization of [9, Lem. 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. *$\left(F_{/\approx}\right)_{/\approx'} \simeq F_{/\approx''}$, with $\approx$ an equivalence relation on F, $\approx'$ an equivalence relation on $F_{/\approx}$, and $\approx''$ the equivalence relation on F obtained by composing $\approx$ and $\approx'$.*
7. *$(F + G)_{/\approx} \simeq F_{/\approx'}$ when G has no edge and all vertices labeled with 1, $\approx$ is an equivalence relation on $F + G$, $\approx'$ is its restriction to F, and for all $x : G$ there exists $y : F$ with $\mathsf{in}_r\, x \approx \mathsf{in}_l\, y$.*

These lemmas allow us to extrude quotients out of unions and to simplify quotients [9]. The generalization to vertex-labeled graphs requires us to verify that vertex labels are collected in a consistent way by the quotient operation (Definition 4.6). While we use ssreflect notations for the 'bigops' operation appearing in this definition, we have to reprove various laws since we only have a monoid structure up to a setoid equality. The generalization to edge-flipping isomorphisms is harmless: the concrete isomorphisms (the first, third, and last three items) do not use this opportunity, and the remaining congruence properties just forward this piece of information.

In addition, we prove a series of basic isomorphisms about interactions between the various operations. We list a few of them below. Together with Lemma 4.11 these properties allow us to reason algebraically and compositionally when we have to prove isomorphisms between graphs. The situation is however not as satisfactory as with standard equational reasoning: the type dependency in congruence lemmas (e.g., Lemma 4.10) prevents us from using standard tools like 'setoid rewriting' so that we generally have to apply these congruence lemmas manually.

Lemma 4.12. *For all graphs F, G, we have*

- *$F \dotplus [x, u, y] \simeq F \dotplus [y, \upsilon, x]$ whenever $u \equiv' \upsilon$,*
- *$F \dotplus [x, u, y] \dotplus [z, \upsilon, t] \simeq F \dotplus [z, \upsilon, t] \dotplus [x, u, y]$,*
- *$F \dotplus [x, u, y][z \leftarrow \alpha] \simeq F[z \leftarrow \alpha] \dotplus [x, u, y]$,*
- *$F \dotplus [x, u, y] + G \simeq (F + G) \dotplus [\mathsf{in}_l\, x, u, \mathsf{in}_l\, y]$,*
- *$(F \dotplus [x, u, y])_{/\approx} \simeq F_{/\approx} \dotplus [\pi x, u, \pi y]$.*

We also prove the following lemma, allowing us to provide an explicit representation for a given quotient graph. We shall see in Section 7 that this makes it possible to prove some concrete isomorphisms by reasoning globally at places where the compositional approach is less convenient.

Lemma 4.13. *Let $G = \langle V, E, p, l^{\upsilon}, l^{e}\rangle$ and $H = \langle V', E', p', l^{\upsilon\prime}, l^{e\prime}\rangle$ be graphs; let $\approx$ be an equivalence relation on V. Let h^{υ} be a surjection from V to V'. Let h^{e} be a bijection from E to E'. If*

1. *for all $e : E$, $b : \mathbb{B}$, $p'\,b\,(h^{e}\ e) = h^{\upsilon}(p\,(h^{d}\ e \oplus b)\ e)$;*
2. *for all $y : V'$, $l^{\upsilon\prime}\ y \equiv \bigotimes_{y = h^{\upsilon}\ x} l^{\upsilon}\ x$;*
3. *for all $e : E$, $l^{e\prime}(h^{e}\ e) \equiv l^{e}\ e$.*

Then $G_{/\approx} \simeq H$.

(The three conditions are almost those of being a homomorphism, except that the condition on vertices labels accounts for the fact that h^{υ} is not injective.)

5 The 2pdom-algebra of graphs

In order to obtain a Σ-algebra, we need to work with two-pointed graphs:

Definition 5.1. A *two-pointed graph* (or *2p-graph* for short) is a structure $\langle G, \iota, o\rangle$ where G is a graph 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. We write $F \simeq_2 G$

$$\langle G,\iota,o\rangle^\circ := \langle G,o,\iota\rangle \quad 1 := \langle 1_1,*,*\rangle \quad \mathrm{dom}(\langle G,\iota,o\rangle) := \langle G,\iota,\iota\rangle$$

$$\langle G,\iota,o\rangle \cdot \langle G',\iota',o'\rangle := \langle (G+G')_{/\approx}, \pi(\mathsf{in}_l\,\iota), \pi(\mathsf{in}_r\,o')\rangle \quad \text{where } \approx := \{(\mathsf{in}_l\,o, \mathsf{in}_r\,\iota')\}^{\mathrm{eqv}}$$

$$\langle G,\iota,o\rangle \parallel \langle G',\iota',o'\rangle := \langle (G+G')_{/\approx}, \pi(\mathsf{in}_l\,\iota), \pi(\mathsf{in}_r\,o')\rangle \quad \text{where } \approx := \{(\mathsf{in}_l\,\iota, \mathsf{in}_r\,\iota'), (\mathsf{in}_l\,o, \mathsf{in}_r\,o')\}^{\mathrm{eqv}}$$

Figure 3. Σ-operations on 2p-graphs.

when two 2p-graphs F and G are isomorphic. The local operations are extended to 2p-graphs in the obvious way, and we overload the notations for these operations.

We can now give a formal definition for the Σ-operations on 2p-graphs (Fig. 3, where R^{eqv} is the equivalence closure of a binary relation R). This is the same definition as in [7, 9] except that it is generalized to graphs over an arbitrary label structure; the way vertex labels are handled is completely hidden in the definition of the quotient operation on graphs (Definition 4.6). Accordingly, since we have generalized [9, Lem. 6.6] to L-labeled graphs (Lemma 4.11), the formal proof [9] of Proposition 3.1 smoothly scales into a proof of the following generalization:

Proposition 5.2. *For every label structure L, 2p-graphs over L form a 2pdom-algebra.*

We finally define the function g interpreting terms into graphs. There are actually two such functions:

- the one from Theorem 3.2, which interpret a term as a letter-labeled graph, i.e., a graph over the label structure from Definition 4.2. This is the unique Σ-homomorphism $\mathrm{g}^{\mathcal{A}}$ such that $\mathrm{g}^{\mathcal{A}}(a) = a^*_*$ for all letters $a \in \mathcal{A}$, where $* : \mathtt{unit}$ is the only allowed vertex-label.
- the one used in the reducibility statement for the rewrite system, which produces a term-labeled graph, i.e., a graph over the label structure from Definition 4.3 applied to the 2pdom-algebra of terms. This is the unique Σ-homomorphism $\mathrm{g}^{\mathcal{T}}$ such that $\mathrm{g}^{\mathcal{T}}(a) = \hat{a}^{\hat{1}}_{\hat{1}}$ for all letters $a \in \mathcal{A}$, where $\hat{1}$ is the term 1 seen as a test, and $\hat{a}$ is the letter a seen as a term.

Since we proved that both algebras of graphs are 2pdom-algebras (Proposition 5.2), we get that $u \equiv v$ entails $\mathrm{g}(u) \simeq_2 \mathrm{g}(v)$ for both functions. Instead, the completeness proof we sketched in Section 3 only gives us a proof of completeness w.r.t. $\mathrm{g}^{\mathcal{T}}$. To get completeness w.r.t. $\mathrm{g}^{\mathcal{A}}$, we build on the following lemma:

Lemma 5.3. *If* $\mathrm{g}^{\mathcal{A}}(u) \simeq_2 \mathrm{g}^{\mathcal{A}}(v)$ *then* $\mathrm{g}^{\mathcal{T}}(u) \simeq_2 \mathrm{g}^{\mathcal{T}}(v)$.

Proof. Follows from relabelings between two label structures being Σ-homomorphisms and preserving isomorphisms. □

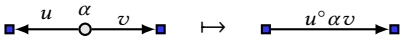
We only work with $\mathrm{g}^{\mathcal{T}}$ in the sequel; we abbreviate it as g.

6 Rewrite System

We now present the graph rewrite system we use to establish completeness of the axioms in Fig. 1. The rewrite system rewrites 2p-graphs labeled with elements of an arbitrary 2pdom-algebra X[1] which we fix in this section.

An informal description the rules is given in Fig. 4. A key intuition about these rules, which does not need to be formalized, is the following. First observe that if X is itself an algebra of 'basic' graphs, so that we have graphs labeled with basic graphs, then one can *expand* a graph into a basic graph by 'replacing' the edges and vertices by their labels. (In fact, the construction of the previous section is a monad in the category of 2pdom-algebras, where multiplication is given by expansion.) The key intuition is that isomorphisms and rewrite rules preserve expansions: if $F \simeq_2 G$ or $F \mapsto G$ then the expansions of F and G are isomorphic (as basic graphs)[2]. This is consistent with the analogy given in Section 3 about state-removal procedures for extracting regular expressions out of finite automata: there, the automata are rewritten through local operations that preserve the overall language.

It is important here that we work modulo edge-flipping isomorphisms. For instance, the following step is an instance of rule (V2) modulo such isomorphisms:



If we were not doing so, we would need two variants of rule (V2) (the one above and the symmetric one), as well as a variant of rule (V1) and a variant of rule (E2). We would thus move from five to nine cases in each case analysis on steps, and from 15 to 45 cases in the local confluence proof. Alternatively, we could add a rule to flip an arbitrary edge. But such a rule is not terminating so that it would prevent us from using Newman's lemma for proving confluence. (All other rules remove either an edge or a vertex, so that the system we use is obviously terminating.)

Remark 6.1. Rule (V0) deserves some explanation, as it is not included in [8]. Note that graphs of the shape $g(u)$ are always connected and that the rewrite system never disconnects connected graphs. Consequently, rule (V0) will never apply when starting from the graph of a term. We include the rule because it makes the rewrite system confluent also on possibly disconnected graphs, without changing its behavior on the connected graphs we care about. This allows us to avoid talking about connectivity, at the cost of five easy additional cases in the confluence proof.

Unlike for the construction of the 2pdom-algebra of graphs, which mainly uses global operations on graphs, the rewrite system is defined using various local operations. There are

[1] Through the label structure from Definition 4.3.

[2] The only exception to this intuition is the rule (V0), which is used in a special way in the present development; see Remark 6.1 below.

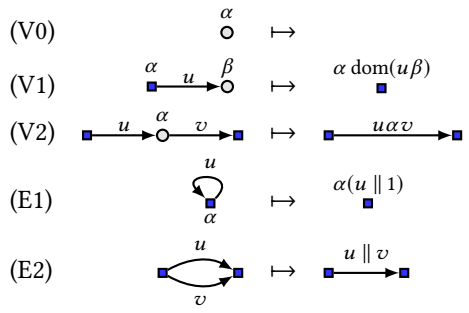


Figure 4. Rewrite rules for term-labeled graphs. The square vertices may have additional incident edges. The circular vertices (i.e., those that are removed) must be distinct from input and output and must not have other incident edges.

moreover at least two fundamentally different ways to formalize this rewrite system: a subtractive version and an additive one. Assume an operation $G \setminus x$ that removes a vertex $x : G$ (and any incident edges) from the graph G. Then rule (V0) can be expressed as rewriting G to $G \setminus z$, provided that $z \notin \{\iota, o\}$ and that z has no incident edges. Alternatively, this rule can be expressed as rewriting $G \dotplus \alpha$ to G. Note that the latter formulation does not require any side condition: the fact that the new vertex is isolated and distinct from input and output is implicit in definition of $G \dotplus \alpha$. This makes the additive formulation rather appealing, especially with the graph representation employed so far. Indeed, the subtractive formulation not only requires explicit side conditions, but also an operation for deleting vertices, which is painful to account for with this representation: deleting $z : G$ from G is only possible if z is neither input nor output, so that vertex deletion on 2p-graphs must take a proof of this fact as additional argument.[3]

Recall from Section 3 that the two most important properties of the rewrite system are:

(ii) Reducibility of $\mathrm{g}(u)$ into a small irreducible graph;
(iii) Confluence up to isomorphism.

It turns out that (ii) can be proved with reasonable effort using an additive formulation of the rewrite system and the packaged representation of graphs. However, as we will argue in Section 8, a subtractive formulation is much more convenient for (iii) and requires us to temporarily use a second representation of graphs.

We now give the formal definition of the rewrite system we use to prove the reducibility property. The rules are given in Fig. 5. Recall that $G \dotplus \alpha$ is defined as the disjoint union of G and the single-vertex graph 1_α whose only vertex is $*$. Hence, its vertices are $\mathsf{in}_l\, x$ for $x : G$ and $\mathsf{in}_r\, *$.

[3] Another possibility would be to do nothing if the passed vertex is the input or the output. However, this would make the type of vertices of $G \setminus z$ dependent on the value of z and therefore would not solve the problem.

Note that, by design, the rules to not yield a graph property (i.e., steps are not preserved under isomorphisms). Thus, we need to close under both isomorphism and transitivity in order to obtain a well-behaved rewrite system. We do so through the following inductive definition:

$$\frac{F \simeq_2 G}{F \Longmapsto G} \qquad \frac{F \simeq_2 F' \qquad F' \mapsto G \qquad G \Longmapsto H}{F \Longmapsto H}$$

Since $\simeq_2$ is reflexive and transitive, so is $\Longmapsto$.

To obtain reducibility in the following section, we need to show that rewrite steps are preserved by the Σ-operations on 2p-graphs:

Lemma 6.2 (Preservation of steps)**.** *If $G \Longmapsto G'$, then*

1. *$G \parallel H \Longmapsto G' \parallel H$ and $H \parallel G \Longmapsto H \parallel G'$,*
2. *$G \cdot H \Longmapsto G' \cdot H$ and $H \cdot G \Longmapsto H \cdot G'$,*
3. *$G^\circ \Longmapsto G'^\circ$ and $\mathrm{dom}(G) \Longmapsto \mathrm{dom}(G')$.*

Since the operations are already known to preserve isomorphisms (part of Proposition 5.2), it suffices by induction on the rewrite sequence to prove the lemma when $G \mapsto G'$. The last item is straightforward since converse and domain do not modify the (non-pointed) graph of their argument. It moreover suffices to do only one side of the first two items since parallel composition is commutative, and since we can use converse to swap the arguments of sequential compositions (again, by Proposition 5.2).

Since both parallel and sequential composition are defined as a quotient of a disjoint union, we factorize most of the work by proving the following technical lemma:

Lemma 6.3. *Let G, G', H be 2p-graphs and let $\approx$ be an equivalence relation on the vertices of $G + H$ such that vertices of G which are neither input nor output are not equivalent to any other vertex. Let $\approx'$ be the equivalence relation mimicking $\approx$ on $G' + H$. If $G \mapsto G'$, then $\langle (G+H)_{/\approx}, \pi \mathsf{in}_l\, \iota_G, \pi \mathsf{in}_r\, o_H \rangle \Longmapsto \langle (G'+H)_{/\approx'}, \pi \mathsf{in}_l\, \iota_{G'}, \pi \mathsf{in}_r\, o_H \rangle$.*

Proof. The lemma intuitively holds because every vertex that can potentially be removed in G by the given rewrite step to G' cannot be the input or the output. Since the quotient touches at most the input and output of G, such a vertex can still be removed in $\langle (G+H)_{/\approx}, \pi \mathsf{in}_l\, \iota_G, \pi \mathsf{in}_r\, o_H \rangle$.

In practice, this lemma is proved compositionally: after a case analysis on the rule used to derive $G \mapsto G'$, we use the commutation properties established between the local operations and the global ones (Lemma 4.12) in order to rewrite $\langle (G+H)_{/\approx}, \pi \mathsf{in}_l\, \iota_G, \pi \mathsf{in}_r\, o_H \rangle$ into an isomorphic graph for which the same rewrite rule syntactically applies, and we prove that the resulting graph is isomorphic to the expected one by using the commutation properties again. □

Note that the use of $\Longmapsto$ in the lemma above is solely to allow for isomorphism steps around a single proper step; this pattern is occurs at several places in our development

$$\text{(V0)}\quad G\dot{+}\alpha \mapsto G$$
$$\text{(V1)}\quad G\dot{+}\alpha\dot{+}[\mathsf{in}_l\, x, u, \mathsf{in}_r\, *] \mapsto G[x \leftarrow \mathrm{dom}(u\alpha)]$$
$$\text{(V2)}\quad G\dot{+}\alpha\dot{+}[\mathsf{in}_l\, x, u, \mathsf{in}_r\, *]\dot{+}[\mathsf{in}_r\, *, v, \mathsf{in}_l\, y] \mapsto G\dot{+}[x, u\alpha v, y]$$
$$\text{(E1)}\quad G\dot{+}[x, u, x] \mapsto G[x \leftarrow u \parallel 1]$$
$$\text{(E2)}\quad G\dot{+}[x, u, y]\dot{+}[x, v, y] \mapsto G\dot{+}[x, u \parallel v, y]$$

Figure 5. Additive presentation of the rewrite system.

7 Reducibility

We work in this section with the 2pdom-algebra of terms over $\mathcal{A}$, and with 2p-graphs labeled using this algebra.

To get a formal reducibility statement, we need to define the normalization function discussed in Section 3, as well as the function g' from normal terms to graphs.

A *normal term* is either a test (α), or a triple (α, u, β) where u is intuitively not a test, although we do not need to keep track of this information. We get small irreducible graphs from normal terms by setting $\mathrm{g}'(\alpha) := 1_\alpha$ and $\mathrm{g}'(\alpha, u, \beta) := u^\beta_\alpha$. We recover a term from a normal term by setting $\lfloor(\alpha)\rfloor := \alpha$ and $\lfloor(\alpha, u, \beta)\rfloor := \alpha u \beta$.

The role of the normalization function $_{\downarrow}$ consists in recursively isolating the tests appearing in a given term. It is defined as the unique Σ-homomorphism such that $a_\downarrow = (1, a, 1)$, after having defined a Σ-algebra on normal terms. For instance, we set

$$(\alpha, u, \beta) \cdot (\gamma, v, \delta) := (\alpha, u\beta\gamma v, \delta)$$
$$(\alpha) \parallel (\beta) := (\alpha\beta)$$
$$(\alpha, u, \beta) \parallel (\gamma, v, \delta) := (\alpha\gamma, u \parallel v, \beta\delta)$$
$$(\alpha, u, \beta) \parallel (\gamma) := (\alpha u \beta \parallel \gamma)$$
$$\mathrm{dom}((\alpha, u, \beta)) := (\alpha\mathrm{dom}(u\beta))$$

(see [8, Fig. 5] for the other cases.)

As explained in Section 3, this syntactic normalization function is valid w.r.t. 2pdom axioms.

Proposition 7.1 (Normalization)**.** *For all terms u, $\lfloor u_\downarrow \rfloor \equiv u$.*

Proof. This amounts to proving that the above equations defining the Σ-algebra on normal terms are all derivable from 2pdom axioms after applying $\lfloor_\rfloor$ on both sides. □

Formalizing reducibility is more challenging:

Proposition 7.2 (Reducibility)**.** *For all terms u, $\mathrm{g}(u) \rightrightarrows \mathrm{g}'(u_\downarrow)$.*

Proof. We proceed by induction on u. In each case, we obtain reduction sequences by induction, which we can combine using the preservation lemma; then it suffices to do a case analysis on the shape of the normal terms, and to perform a last rewrite step if necessary.

Suppose for instance that $u = v \parallel w$. By induction we get reduction sequences $\mathrm{g}(v) \rightrightarrows \mathrm{g}'(v_\downarrow)$ and $\mathrm{g}(w) \rightrightarrows \mathrm{g}'(w_\downarrow)$, from which we deduce by Lemma 6.2 that $\mathrm{g}(u) = \mathrm{g}(u) \parallel \mathrm{g}(v) \rightrightarrows \mathrm{g}'(v_\downarrow) \parallel \mathrm{g}'(w_\downarrow)$. Then there are three cases to consider:

- either both $v_\downarrow$ and $w_\downarrow$ are tests, say (α) and (β), so that $u_\downarrow = (\alpha\beta)$. In this case it suffices to check that
$$\mathrm{g}'(v_\downarrow) \parallel \mathrm{g}'(w_\downarrow) = 1_\alpha \parallel 1_\beta \simeq_2 1_{\alpha\beta} = \mathrm{g}'(u_\downarrow)$$
- or none of them is a test: $v_\downarrow = (\alpha, v', \beta)$, $w_\downarrow = (\gamma, w', \delta)$, $u_\downarrow = (\alpha\gamma, v' \parallel w', \beta\delta)$, and we check that
$$\begin{aligned}\mathrm{g}'(v_\downarrow) \parallel \mathrm{g}'(w_\downarrow) &= {v'}^\beta_\alpha \parallel {w'}^\delta_\gamma \\ &\simeq_2 2^{\beta\delta}_{\alpha\gamma} \dot{+} [\mathsf{in}_l\, *, v', \mathsf{in}_r\, *] \dot{+} [\mathsf{in}_l\, *, w', \mathsf{in}_r\, *] \\ &\mapsto 2^{\beta\delta}_{\alpha\gamma} \dot{+} [\mathsf{in}_l\, *, v' \parallel w', \mathsf{in}_r\, *] \qquad \text{(E2)} \\ &= (v' \parallel w')^{\beta\delta}_{\alpha\gamma} = \mathrm{g}'(u_\downarrow)\end{aligned}$$
- or one of them is a test and the other is not, say $v_\downarrow = (\alpha, v', \beta)$ and $w_\downarrow = (\gamma)$, so that $u_\downarrow = (\alpha v' \beta \parallel \gamma)$. There we have
$$\begin{aligned}\mathrm{g}'(v_\downarrow) \parallel \mathrm{g}'(w_\downarrow) &= {v'}^\beta_\alpha \parallel 1_\gamma \\ &\simeq_2 1_{\alpha\beta\gamma} \dot{+} [*, v', *] \\ &\mapsto 1_{\alpha\beta\gamma(v' \parallel 1)} \qquad \text{(E0)} \\ &\simeq_2 1_{\alpha v'\beta \parallel \gamma} = \mathrm{g}'(u_\downarrow)\end{aligned}$$
The last isomorphism comes from the law $\alpha\beta\gamma(v' \parallel 1) \equiv \alpha v'\beta \parallel \gamma$, which is indeed derivable.

In those three cases for parallel composition, the first isomorphism we have to provide is between the graph $\mathrm{g}'(v_\downarrow) \parallel \mathrm{g}'(w_\downarrow)$, which is a quotient of a graph with two to four vertices, and a concrete graph with one or two vertices. We could prove those isomorphisms compositionally, but it turned out to be more convenient to prove them using Lemma 4.13, by providing surjective homomorphisms explicitly.

The other Σ-operations are handled similarly, domain and converse of course being simpler. □

8 Confluence

We use Newman's lemma to prove confluence of $\rightrightarrows$ via local confluence of the single step relation $\mapsto$. Since we work modulo isomorphisms and our single step relation is not a graph property, the appropriate local confluence property is the following one:

Proposition 8.1 (Local confluence modulo isomorphisms)**.** *If $F' \leftarrow\!\!\shortmid F \simeq_2 G \mapsto G'$, then there exists H such that $F', G' \rightrightarrows H$.*

Thanks to the inductive definition we used for $\rightrightarrows$, we easily formalize the required variant of Newman's lemma to deduce confluence of $\rightrightarrows$.

Formalizing the above local confluence lemma however turns out to be extremely tedious using packaged graphs and our additive definition of the rewrite rules. To see why, consider one of the simplest cases, the interaction of two instances of the (V1) rule. Our assumptions in this case are:

- $F \dot{+} \alpha \dot{+} [\mathsf{in}_l\, x, u, \mathsf{in}_r\, *] \simeq_2 G \dot{+} \beta \dot{+} [\mathsf{in}_l\, y, v, \mathsf{in}_r\, *]$

- $F \dotplus \alpha \dotplus [\mathsf{in}_l\, x, u, \mathsf{in}_r\, *] \mapsto F[x \leftarrow \mathrm{dom}(u\alpha)]$
- $G \dotplus \beta \dotplus [\mathsf{in}_l\, y, v, \mathsf{in}_r\, *] \mapsto G[y \leftarrow \mathrm{dom}(v\alpha)]$

In order to close this pair, we first have to check whether the isomorphism maps $\mathsf{in}_r\, *$ to $\mathsf{in}_r\, *$ (in which case the two instances are the same). Otherwise, we need to trace the vertex removed on the right through the isomorphism in order to expand F accordingly and vice versa. Moreover, one needs to exhibit an isomorphism between the untouched parts of F and G. This happens in a dependently typed setting and our attempts to carry out these constructions failed due to circular dependencies between the involved statements. Moreover, even if this problem could be solved, it would not lead to a natural proof of local confluence: the vertices of interest each have two names (in F and in G), keeping track of their correspondences rapidly grows out of hands, and we cannot organize case distinctions in the most efficient way.

Instead, we prove local confluence using an alternative representation of both graphs and the step relation. As it comes to graphs, we employ *open graphs*: graphs where both vertices and edges are just natural numbers. Although such a formal representation is commonplace [1, 13–15, 17], we are not aware of any work where a correspondence with a packaged representation was established and exploited. For the step relation, we employ a subtractive characterization such that single steps can be performed independently from the concrete shape of the considered graph.

Definition 8.2 (Open graph)**.** We fix two copies $\mathcal{V}$ and $\mathcal{E}$ of the countably infinite type $\mathbb{N}$ of natural numbers and call them *vertices* and *edges* respectively. An *(open) pre-graph* is a tuple $U = \langle V_U, E_U, p_U, l^v_U, l^e_U, \iota_U, o_U\rangle$ where

- V_U is a finite set of vertices
- E_U is a finite set of edges
- $p_U : \mathbb{B} \to \mathcal{E} \to \mathcal{V}$ indicates the source and target of edges
- $l^v_U : \mathcal{V} \to L_v$ indicates the labels of vertices
- $l^e_U : \mathcal{E} \to L_e$ indicates the labels of edges
- ι_U and o_U indicate the input and the output

A pre-graph is *well-formed* (i.e, an open graph) if the following conditions hold:

- $p_U\, b\, e \in V_U$ for all $b : \mathbb{B}$ and $e \in E_U$.
- $\iota_U, o_U \in V_U$.

We let $O, U, V \ldots$ range over open pre-graphs. The separation into pre-graphs and well-formedness is crucial here. It allows us to separate the task of proving that a graph has the desired shape from proving that it is actually a graph.

In Coq, we use the finmap library [3] to represent finite sets over the countable types $\mathcal{V}$ and $\mathcal{E}$. Moreover, we turn the well-formedness predicate into a class, allowing it to be inferred it automatically in many situations.

We then define operations corresponding to those defined on 2p-graphs. In the case of open graphs, where edges and vertices are external, the operations adding vertices and edges take the edge to be added as additional argument. That is, we have the following operations:

- $U \dotplus [x, \alpha]$ denotes U with vertex x labeled with α
- $U \dotplus [e, x, u, y]$ denotes U with e u-labeled edge from x to y.
- $U[x \leftarrow a]$ denotes U where x is labeled with $a \otimes l^v\, U\, x$.
- $U \setminus x$ denotes U with x and all edges incident to x removed.
- $U - E$ denotes U with all the edges in E removed.

All these operations are easily defined as operations on pre-graphs. They yield graphs under the expected side conditions (e.g., adding an edge requires that the vertices are part of the graph and deleting a vertex requires that the deleted vertex is distinct from input and output). In particular, vertex deletion, which we will use extensively below, is a benign operation on open graphs. In contrast, the quotient and disjoint union operations we extensively used in the previous sections would be painful to deal with on open graphs.

We have the following notion of strong equivalence: two pre-graphs are strongly equivalent if they only differ by their labels, up to the setoid relations on L_v and L_e.

Definition 8.3 (Strong equivalence)**.** Let O and U be pre-graphs. We call O and U *strongly equivalent*, written $O \equiv U$, if the following conditions are satisfied

1. $V_O = V_U$ and $E_O = E_U$.
2. the functions p_O and p_U agree on V_O
3. $l^v_O x \equiv l^v_U x$ for all $x \in V_O$
4. $l^e_O e \equiv l^e_U e$ for all $e \in E_O$
5. $\iota_O = \iota_U$ and $o_O = o_U$

In other words, strong equivalences correspond to non-edge flipping isomorphisms which are the identity on both vertices and edges. This relation enjoys a number of useful properties, of which we list only a few:

Lemma 8.4. *1. $\equiv$ is an equivalence relation on pre-graphs*
2. $U[x \leftarrow \alpha][y \leftarrow \beta] \equiv U[y \leftarrow \beta][x \leftarrow \alpha]$
3. $U \setminus x \setminus y = U \setminus y \setminus x$
4. $U \dotplus [e, x, u, y] \setminus z \equiv (U \setminus z) \dotplus [e, x, u, y]$ if $z \notin \{x, y\}$ and $e \notin E_U$
5. $U \dotplus [e, x, u, y] \setminus y \equiv U \setminus y$
6. $U \dotplus [x, a] \setminus x \equiv U$ if U is well-formed and $x \notin V_U$.
7. $U[x \leftarrow \alpha] \equiv V[x \leftarrow \beta]$ if $U \equiv V$ and $\alpha \equiv \beta$
8. $U \dotplus [e, x, u, y] \equiv V \dotplus [e, x, v, y]$ if $U \equiv V$ and $u \equiv v$

Note that most of these equivalences hold irrespective of whether the graph G is well-formed or not. A notable exception is (6), where the addition of x to a graph that is not well-formed could turn a formerly "dangling" edge into one that is incident to x and would therefore be removed by the deletion of x. That the last two items hold is extremely convenient as it allows us to use 'setoid rewriting', something we cannot use for isomorphisms on packaged graphs due to the dependency between the replaced graphs and their vertices (cf. Lemma 4.10).

$$\text{(V0)}\ \frac{z \in V_U \qquad I_U(z) = \emptyset \qquad z \notin IO_U}{U \mapsto U \setminus z}$$

$$\text{(V1)}\ \frac{I_U(z) = \{e\} \qquad \operatorname{arc} U\, e\, x\, u\, z \qquad x \neq z \qquad z \notin IO_U}{U \mapsto U[x \leftarrow \operatorname{dom}(u \cdot l_U^v\, z)] \setminus z}$$

$$\text{(V2)}\ \frac{\begin{array}{c} I_U(z) = \{e_1, e_2\} \qquad e_1 \neq e_2 \qquad z \notin \{x, y\} \\ z \notin IO_U \qquad \operatorname{arc} U\, e_1\, x\, u\, z \qquad \operatorname{arc} U\, e_2\, z\, v\, y \end{array}}{U \mapsto U \setminus z \dotplus [\max(e_1, e_2), x, u \cdot l_U^v\, z \cdot v, y]}$$

$$\text{(E1)}\ \frac{\operatorname{arc} U\, e\, x\, _\, x}{U \mapsto (U - \{e\})[x \leftarrow l_U^e\, e]}$$

$$\text{(E2)}\ \frac{\operatorname{arc} U\, e_1\, x\, u\, y \qquad \operatorname{arc} U\, e_2\, x\, v\, y \qquad e_1 \neq e_2}{U \mapsto (U - \{e_1, e_2\}) \dotplus [\max(e_1, e_2), x, u \parallel v, y]}$$

Figure 6. Open step relation.

We now turn to the definition of the step relation. We want steps to be preserved under isomorphism so that the local confluence property can be stated using a single graph as a left-hand side of two reduction steps. This forces us to handle the edge-flipping behavior in the definition of the rules. To this end, we define a predicate expressing whether an edge can, up to reversal, be seen as a given edge, and then use this predicate in the definition of the step relation.

$$\operatorname{arc} U\, e\, x\, u\, y :=$$
$$e \in E_U \wedge \exists b.\, p_U\, b\, e = x \wedge p_U\, (\neg b)\, e = y \wedge l_U^e\, e \equiv [b]\, u$$

We moreover write $I_G(x)$ for the set of edges incident to x in G. The rules of our subtractive variant of the step relation are given in Fig. 6. We use a max operation on edges (natural numbers) in rules (V2) and (E2): by doing so, the name of the edge which is kept depends only on the edges being removed and not on the order in which the edges are matched. This is convenient in that it allows us to close local confluence critical pairs using strong equivalences rather than isomorphisms.

We define the multi-step rewrite relation $\mapsto\!\!\!\!\!\Rightarrow$ as the least transitive relation containing strong equivalence, single steps, but also the relation that flips a single edge in a graph. This is fine since we reason modulo edge-flipping isomorphisms on packaged graphs; doing so makes it possible to use symmetry reasoning in the analysis of the critical pairs.

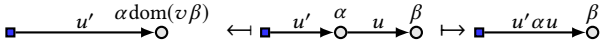
Now we have everything in place to prove local confluence

Proposition 8.5. *Let O, U, V be open graphs such that $U \mathrel{\dot{\leftarrow}} O \mathrel{\dot{\mapsto}} V$. Then there exist graphs U' and V' such that $U \mapsto\!\!\!\!\!\Rightarrow U'$, $V \mapsto\!\!\!\!\!\Rightarrow V'$ and $U' \equiv V'$.*

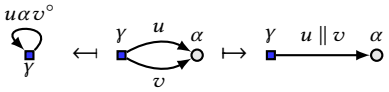
Since $\mapsto\!\!\!\!\!\Rightarrow$ is closed under strong equivalence, the statement is equivalent to the one where U' is syntactically equal to V'. We prefer this formulation because it matches the way we prove the lemma.

Proof. The proof boils down to a case distinction between all the possible pairs of rules that could have been applied to obtain $U \mathrel{\dot{\leftarrow}} O \mathrel{\dot{\mapsto}} V$. We can reduce the number of cases by assuming, without loss of generality, that the rule applied on the left has a lower index (with respect to the textual order in Fig. 6). In Coq, this is realized by defining the relation $\mathrel{\dot{\mapsto}}$ as a relation in Type and defining a function numbering the rules. There are still fifteen cases left, and each of them comes with its own case distinctions. We discuss only some illustrative cases below.

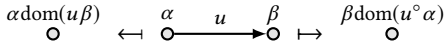
- Consider an interaction of the rule (V1) with $\operatorname{arc} O\, e\, x\, u\, z$ and rule (V2) with $\operatorname{arc} O\, e_1\, x'\, u'\, z'$ and $\operatorname{arc} O\, e_2\, z'\, v'\, y'$. We have $z \neq z'$ due to the different number of incident edges. If $z \notin \{x', y'\}$ the instances are independent, so assume $z \in \{x', y'\}$. Without loss of generality, we can assume $z = x'$, for otherwise we exchange the roles of x' and y' as well as e_1 and e_2, setting $u' := v'^{\circ}$ and $v' := u'^{\circ}$ and flipping the edge introduced through the application of (V2). Hence, $e_2 = e$, $z' = x$, and $v' = u$, exhibiting the situation below:



  After applying rule (V1) on both sides and using the equivalences from Lemma 8.4, it suffices to show $\operatorname{dom}(u\alpha v\beta) \equiv \operatorname{dom}(u\alpha\operatorname{dom}(v\beta))$.
- Rules (V2) and (E2) can interact as follows:



  This pair can be joined by applying rule (E1) in the left and rule (V1) on the right, yielding the derivable equation $u\alpha v^{\circ} \parallel 1 \equiv \operatorname{dom}((u \parallel v)\alpha)$.
- Rule (V1) can interact with itself as follows:



  This pair can be joined by applying Rule (V0) on both sides, not requiring any equation. □

Note that we need to allow edge-flips in the definition of $\mapsto\!\!\!\!\!\Rightarrow$ in order to factor the proof by reasoning without loss of generality in the first case discussed in the proof. This is important: we use this pattern thirteen times in the full proof, which is 500 lines long.

Also remark that the last case mentioned above does not arise in [8] where the graphs are assumed to be connected.

9 Transferring Local Confluence

Proposition 8.5 proves local confluence, but not using the representation employed in the rest of the formalization. In

this section, we outline what is needed in order to transfer this result and obtain a proof of Proposition 8.1.

We first need translation functions between the two representations of graphs. We start with the translation from open graphs to packaged graphs.

Definition 9.1 (Packing)**.** Let U be an open graph. We define the *packing of* U, written $\mathsf{pck}\, U$, to be the 2p-graph whose finite type of vertices is the finite type of elements of V_U (also denoted by V_U[4]) and likewise for edges. The remaining components are obtained by casting the components of U to the appropriate types (e.g., $p : \mathbb{B} \to E_U \to V_U$ for the endpoints of edges).

Since finite sets over countable types coerce to finite types in Coq, this packing operation is easy to define.

For the converse translation, from packaged graphs to open graphs, the main issue is in embedding the finite types of edges and vertices of a given packaged graph into the generic vertex and edge types. For these, we employ two generic injections

$$\mathsf{inj}^v : \forall T : \mathsf{finType}.\, T \to \mathcal{V}$$
$$\mathsf{inj}^e : \forall T : \mathsf{finType}.\, T \to \mathcal{E}$$

Both of these come only with partial inverses in general, since T may be empty. This is never the case for the vertex type of 2p-graphs however, which must contain input and output, so that we get:

$$\mathsf{proj}^v : \forall G : \text{2p-graph}.\, \mathcal{V} \to \mathsf{vertex}\, G$$
$$\mathsf{proj}^e : \forall G : \text{2p-graph}.\, \mathcal{E} \to (\mathsf{edge}\, G)_\perp$$

The functions are defined such that $\mathsf{proj}^v(\mathsf{inj}^v v) = v$ whenever v is a vertex (of some 2p-graph) and $\mathsf{proj}^e(\mathsf{inj}^e e) = \mathsf{Some}\, e$ whenever e is an edge. The operation of opening a packaged graph can then be defined as follows

Definition 9.2 (Opening)**.** Let $G = \langle V, E, p, l^v, l^e, \iota, o\rangle$. We set $\mathsf{opn}\, G := \langle V', E', p', l^{v\prime}, l^{e\prime}, \mathsf{inj}^v \iota, \mathsf{inj}^v o\rangle$ with

- $V' := \{\mathsf{inj}^v x \mid x : G\}$
- $E' := \{\mathsf{inj}^e e \mid e : \mathsf{edge}\ G\}$
- $p'\, b\, e := \mathsf{if}\ \mathsf{proj}^{\mathsf{e}}\, e\ \mathsf{is\ Some}\, e'\ \mathsf{then}\ \mathsf{inj}^v(p\, b\, e')\ \mathsf{else}\ \iota$
- $l^{v\prime} v := l^v(\mathsf{proj}^v v)$
- $l^{e\prime} e := \mathsf{if}\ \mathsf{proj}^{\mathsf{e}}\, e\ \mathsf{is\ Some}\, e'\ \mathsf{then}\ l^e e'\ \mathsf{else}\ 1$

At this point, we can prove:

Lemma 9.3. *For all 2p-graph* G, $\mathsf{pck}(\mathsf{opn}\, G) \simeq_2 G$.

In order to relate the two variants of the step relation, we need a proper notion of isomorphism on open graphs. This is obtained by reusing isomorphisms on packaged graphs via the packing operation.

Definition 9.4 (Open isomorphisms)**.** Let O and U be open pre-graphs. We call O and U isomorphic, written $O \doteq_2 U$ if they are both well-formed and $\mathsf{pck}\, O \simeq_2 \mathsf{pck}\, U$.

[4]This is the finite type of pairs of elements $x : \mathcal{V}$ and proofs of $x \in V_U$.

Note that we define isomorphism as a relation between pre-graphs; this relation is a partial equivalence relation (PER) rather than an equivalence relation due to the well-formedness requirement.

As expected, strong equivalences give rise to isomorphisms.

Fact 9.5. *$O \doteq_2 U$ whenever $O \equiv U$ and O is well-formed.*

Note that, given $O \equiv U$, well-formedness of O implies well-formedness of U, allowing us to have only one well-formedness assumption. This is convenient when showing that a complex graph expression is isomorphic to an expression known to be well-formed.

Now that we have a proper notion of isomorphisms on open graphs, we can show that the open step relation commutes with them.

Lemma 9.6. *If $U \mapsto U'$ and $U \doteq_2 V$, then there exists some V' such that $V \mapsto V'$ and $U' \doteq_2 V'$.*

This property is relatively easy to prove because it deals exclusively with open graphs. All we need to do is establish a number of lemmas showing that incidence of edges and the arc predicate are preserved under isomorphisms. As with isomorphisms of packaged graphs, this requires viewing isomorphisms as functions from vertices to vertices.

We then need to prove that (additive) packaged steps and (subtractive) open step actually match. To prove that packaged steps give rise to open steps we first establish a number of commutation properties about the opening operation:

Fact 9.7.
- $\mathsf{opn}(G[x \leftarrow \alpha]) \doteq_2 (\mathsf{opn}\, G)[\mathsf{inj}^v x \leftarrow \alpha]$
- $\mathsf{opn}(G \dotplus \alpha) \doteq_2 \mathsf{opn}\, G \dotplus [x, \alpha]$ *when* $x \notin E_{\mathsf{opn}\, G}$
- $\mathsf{opn}(G \dotplus [x, u, y]) \doteq_2 \mathsf{opn}\, G \dotplus [e, \mathsf{inj}^v x, u, \mathsf{inj}^v y]$ *when* $e \notin V_{\mathsf{opn}\, G}$

Lemma 9.8. *If $F \mapsto F'$ then there exists some U' such that* $\mathsf{opn}\, F \mapsto U'$ *and* $\mathsf{pck}\, U' \simeq_2 F'$.

Proof. By case analysis on $F \mapsto F'$. We sketch the case for the (V1) rule, the other cases are similar. That is, we need to find a graph U' such that

$$\mathsf{opn}(G \dotplus \alpha \dotplus [\mathsf{in}_l\, x, u, \mathsf{in}_r\, *]) \mapsto U' \tag{1}$$
$$U' \doteq_2 \mathsf{opn}(G[x \leftarrow \mathsf{dom}(u\alpha)]) \tag{2}$$

By Lemma 9.6, we can also make isomorphism steps before making an actual step. We pick a fresh vertex z and a fresh edge e and then use Fact 9.7 to push the opening operation down to the graph G. Thus (1) reduces to showing:

$$H := \mathsf{opn}\, G \dotplus [z, \alpha] \dotplus [e, \mathsf{inj}^v\, x, u, z] \mapsto U'$$

The side conditions of rule (V1) are then easily established. Applying the rule (V1) determines U' to be

$$H[\mathsf{inj}^v\, x \leftarrow \mathsf{dom}(u\alpha)] \setminus z$$

whose packing is isomorphic to $G[x \leftarrow \mathsf{dom}(u\alpha)]$. □

For the converse direction (reflecting open steps with packaged steps), we need to be able to trace vertices through the packing operation. For this, we define a function

$$\mathsf{pck}^{v} : \forall U.\, \mathcal{V} \to U$$

Recall that the vertex type of $\mathsf{pck}\, U$ is the set of vertices of U. Hence, this amounts to pairing vertices x with proofs of $x \in U$ if possible and otherwise returning a default vertex.

Fact 9.9.
- $\mathsf{pck}\,(U[x \leftarrow \alpha]) \simeq_2 (\mathsf{pck}\, U)[\mathsf{pck}^{v}\, x \leftarrow \alpha]$
- $\mathsf{pck}\,(U \dotplus [x, \alpha]) \simeq_2 \mathsf{pck}\, U \dotplus \alpha$ *when* $x \notin V_U$
- $\mathsf{pck}(U \dotplus [e, x, u, y]) \simeq_2 \mathsf{pck}\, U \dotplus [\mathsf{pck}^{v} x, u, \mathsf{pck}^{v} y]$ *when* $e \notin E_U$

Lemma 9.10. *If* $U \dot{\Longrightarrow} U'$, *then* $\mathsf{pck}\, U \Longrightarrow \mathsf{pck}\, U'$.

Proof. By induction on the given sequence, since strong equivalence and edge-flips are both contained in isomorphism, it suffices to prove the lemma in the case of a single rewrite step. We proceed by case analysis on this step. We again sketch the case for (V1), the other cases are similar. We have $I_G(z) = \{e\}$, $\mathsf{arc}\, U\, e\, x\, u\, z$, $z \neq x$, and $z \notin \{\iota_U, o_U\}$ and we need to show

$$\mathsf{pck}\, U \Longrightarrow \mathsf{pck}(U[x \leftarrow \mathrm{dom}(u\alpha)] \setminus z)$$

where $\alpha := l^{v}_{U}\, z$. In order to apply the (V1) rule for $\mapsto$, we need to expand U. Without loss of generality, e is an edge from x to z; if not, we apply an isomorphism that reverses e in U and reestablish all assumptions. Now it suffices to show

$$\mathsf{pck}\, U \simeq_2 \mathsf{pck}(U \setminus z) \dotplus \alpha \dotplus [\mathsf{in}_l\,(\mathsf{pack}^{v}\, x), u, \mathsf{in}_r\, *] \qquad (3)$$

and apply rule (V1). Finally, the isomorphism (3) is established by extruding the packing operation on the right. During this process, we can choose the vertex and edge to be added (Fact 9.9) and we choose x and e respectively. Thus, it suffices to show

$$\mathsf{pck}\, U \simeq_2 \mathsf{pck}(U \setminus z \dotplus [z, \alpha] \dotplus [e, x, u, z])$$

which can be shown using the laws for graph equivalence (Fact 9.5 and Lemma 8.4). □

We can finally transfer local confluence:

Proof of Proposition 8.1. Assume $F' \leftarrowtail F \simeq_2 G \mapsto G'$. By Lemma 9.8, we get U', V' such that $\mathsf{opn}\, F \dot{\mapsto} U'$, $\mathsf{pck}\, U' \simeq_2 F'$, $\mathsf{opn}\, G \dot{\mapsto} V'$, and $\mathsf{pck}\, V' \simeq_2 G'$. By Lemma 9.3, we have $\mathsf{opn}\, F \doteq_2 \mathsf{opn}\, G$, so that by Lemma 9.6 we get W' such that $\mathsf{opn}\, G \dot{\mapsto} W'$ and $U' \doteq_2 W'$. By open local confluence (Proposition 8.5) on $\mathsf{opn}\, G$, we finally find O such that $V', W' \dot{\Longrightarrow} O$. We can close the diagram with $\mathsf{pck}\, O$: by Lemma 9.10 we have

$$F' \simeq_2 \mathsf{pck}\, U' \simeq_2 \mathsf{pck}\, W' \Longrightarrow \mathsf{pck}\, O \Longleftarrow \mathsf{pck}\, V' \simeq_2 G' \qquad \square$$

10 Conclusion and future work

Putting everything together as explained at the end of Section 3, we obtain a formal proof Theorem 3.2. Formalizing this proof required us to setup several tools for reasoning about graphs and isomorphisms, under two distinct representations: the packaged representation, which is convenient for constructing and combining graphs, and the open representation, which is convenient for subtractive operations.

The formalization has a moderate size (2700/3900 lines of specification/proofs); large parts of it consist in general infrastructure and lemmas about labeled directed multigraphs, which should be reusable and could be integrated with [9].

We leave four main directions for future work.

First, building on the finite type library of MathComp [18] is extremely convenient, but the few additional structures we added on top of it in a monolithic way (labels, graphs, 2p-graphs) already made us reach the point where efficiency issues start being noticeable. We thus plan to rework the concrete implementation of these structures, following the packaged-class approach advocated in MathComp [12]. This should improve efficiency, but also give us the opportunity to provide a library with more fine-grained abstractions.

Second, 2pdom-algebras form a fragment of 2p-algebras, where a neutral element for parallel composition is added, allowing one to denote all graphs of treewidth at most two—not just the connected ones. Soundness and completeness of 2p-algebra is established in [4, 8], each time by building on completeness of 2pdom-algebra. We plan to explore a more direct approach: we believe that the present formal proof could be extended to deal directly with this more general case, just by modifying the rewrite system and the shape of the normal terms. We would also like to study smaller fragments. For instance, is it still possible to axiomatize graph isomorphism in absence of the converse operation?

Third, by combining our completeness proof for 2pdom-algebra with the extraction function from treewidth at most two connected graphs to terms [4] formalized in [7], we could obtain a formal proof that such graphs form the free 2pdom-algebra. The rewrite system we used here makes it possible to define another extraction function [8], whose informal description is much easier. Formalizing this approach however seems challenging: it requires to study the expansion function we alluded to at the beginning of Section 6, which is a global and deeply dependently typed operation.

Fourth, completeness of 2pdom-algebra has been recently used to obtain that the equational theory of allegories [11] is decidable [16]. This decidability proof involves more graph theory than the present one: it exploits the fact that the graphs of treewidth at most two are precisely those excluding K_4 as a minor, and it requires a long analysis of the possible homomorphisms occurring within such graphs. Formalizing this proof seems rather challenging but not out of reach, thanks to the library we have developed so far.

References

[1] R. Chen, C. Cohen, J. Lévy, S. Merz, and L. Théry. Formal proofs of tarjan's strongly connected components algorithm in why3, coq and isabelle. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *ITP*, volume 141 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[2] C. Cohen. Pragmatic quotient types in Coq. In *Proc. ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2013.

[3] C. Cohen. A finset and finmap DRAFT library. https://github.com/math-comp/finmap, Nov. 2017. Accessed Nov. 17th, 2017.

[4] E. Cosme-Llópez and D. Pous. k_4-free graphs as a free algebra. In *Proc. MFCS*, volume 83 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[5] 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.

[6] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

[7] 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, *ITP*, volume 10895 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2018.

[8] 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.

[9] C. Doczkal and D. Pous. Graph theory in coq: Minors, treewidth, and isomorphisms, 2019. submitted.

[10] R. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.

[11] P. Freyd and A. Scedrov. *Categories, Allegories*. North Holland. Elsevier, 1990.

[12] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.

[13] A. Guéneau, J. Jourdan, A. Charguéraud, and F. Pottier. Formal proof and analysis of an incremental cycle detection algorithm. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *ITP*, volume 141 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[14] P. Lammich and S. R. Sefidgar. Formalizing network flow algorithms: A refinement approach in isabelle/hol. *Journal of Algebraic Reasoning*, 62(2):261–280, 2019.

[15] L. Noschinski. A graph library for Isabelle. *Mathematics in Computer Science*, 9(1):23–39, 2015.

[16] D. Pous and V. Vignudelli. Allegories: decidability and graph homomorphisms. In *Proc. LiCS*, pages 829–838. ACM, 2018.

[17] A. K. Singh and R. Natarajan. Towards a constructive formalization of perfect graph theorems. In M. A. Khan and A. Manuel, editors, *ICLA*, volume 11600 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2019.

[18] The Mathematical Components team. Mathematical components – libraries of formalized mathematics. http://math-comp.github.io/math-comp/.