Open-graphs and monoidal theories[†]

L U C A S   D I X O N[‡]   and   A L E K S   K I S S I N G E R[§]

[‡]*Google, New York, U.S.A.*
and
University of Edinburgh, Edinburgh, United Kingdom
Email: `lucas.dixon@ed.ac.uk`
[§]*University of Oxford, Oxford, United Kingdom*
Email: `aleks0@gmail.com`

String diagrams are a powerful tool for reasoning about physical processes, logic circuits, tensor networks and many other compositional structures. The distinguishing feature of these diagrams is that edges need not be connected to vertices at both ends, and these unconnected ends can be interpreted as the inputs and outputs of a diagram. In this paper, we give a concrete construction for string diagrams using a special kind of typed graph called an *open-graph*. While the category of open-graphs is not itself adhesive, we introduce the notion of a *selective adhesive functor*, and show that such a functor embeds the category of open-graphs into the ambient adhesive category of typed graphs. Using this functor, the category of open-graphs inherits 'enough adhesivity' from the category of typed graphs to perform double-pushout (DPO) graph rewriting. A salient feature of our theory is that it ensures rewrite systems are 'type safe' in the sense that rewriting respects the inputs and outputs. This formalism lets us safely encode the interesting structure of a computational model, such as evaluation dynamics, with succinct, explicit rewrite rules, while the graphical representation absorbs many of the tedious details. Although topological formalisms exist for string diagrams, our construction is discrete and finitary, and enjoys decidable algorithms for composition and rewriting. We also show how open-graphs can be parameterised by graphical signatures, which are similar to the monoidal signatures of Joyal and Street, and define types for vertices in the diagrammatic language and constraints on how they can be connected. Using typed open-graphs, we can construct free symmetric monoidal categories, PROPs and more general monoidal theories. Thus, open-graphs give us a tool for mechanised reasoning in monoidal categories.
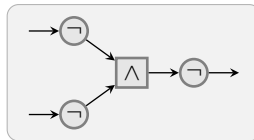
1. Introduction

Graphs are often used for specification and reasoning, both formally and informally. They have both an appealing visual nature as well as the ability to abstract structure naturally. In this paper we will focus on 'string diagrams', which are the graphical structures that arise in monoidal theories. Well-known examples include proof-nets in linear logic (Girard 1996), Penrose's tensor notation (Penrose 1971), Feynman diagrams, diagrammatic notations

---

for logic circuits and high-level languages for quantum information processing (Coecke and Duncan 2008). A common feature of these graphical languages is that they can be understood as describing a computational process, and that they support reasoning by manipulating the graphical presentation. However, such manipulation is both tedious and error prone to do by hand. In this paper, we address this difficulty by providing a generic, but also concrete and computable, account of graphical reasoning in monoidal-theories. Our long-term goal is to support automation for graphical reasoning about computational structures.

The main concept we introduce is a formal theory of *open-graphs*. Like graph-based drawings of circuits, the visual presentation of open-graphs consists of vertices connected by edges. Crucially, edges in an open-graph need not be attached to vertices. They may be unconnected at one or both ends, or even connected to themselves to form a 'circle'. In terms of a computational process, the unconnected ends of edges represent the inputs and outputs of a process. A diagram in this graphical language is interpreted as a compound computation with vertices as the atomic operations and wires defining the flow of information. For example, an electronic circuit that defines the compound logical operation of an or-gate using not-gates around an and-gate can be drawn as

Open-graphs have a rich compositional structure and a convenient algebraic language. We introduce methods for plugging graphs together, merging over common subgraphs and cutting out pieces of a graph. Using these tools, we develop rewriting for open-graphs. In this regard, our formalism functions analogously to a type-system in a programming language: we ensure that the interface of a process is maintained by rewriting. In particular, we show that rewriting also has a compositional nature: the decomposition of graphs by cutting their edges enables rewriting to be performed in parallel on the separated components, with a guarantee that the separate rewritten parts can be recomposed appropriately. Moreover, the compositional properties of open-graphs allow rewrite rules themselves to be rewritten using the same machinery.

To formalise the process of rewriting, we use a well-behaved embedding of the category of open-graphs into its ambient category of typed graphs. This embedding is an instance of a more general notion, which we introduce as *selective adhesive functors*. In particular, these functors reflect pushouts, so many results about pushouts in an adhesive category are true of so-called *adhesive pushouts*, which are the pushouts reflected by a selective adhesive functor.

We also parameterise the category of open-graphs by a *graphical signature*. This defines a collection of vertex and edge types and assigns to each vertex type its input and output types. We construct a *typegraph* from such a signature and form the category of *typed open-graphs* by slicing over this typegraph. Combined with a collection of graphical rules,

these typed open-graphs provide a formal way to reason with a graphical theory of some algebraic or dynamical system. We demonstrate the generality of our construction by showing that typed open-graphs can be used to construct free symmetric monoidal categories, PROPs and a wide range of more general monoidal theories. Unlike many other (topological) constructions for diagrammatic accounts of monoidal categories, our construction involves finite data. Thus our construction enables the development of software tools that work with graphical theories. In particular, it provides the basis for employing techniques from automated reasoning, such as completion-based methods (Knuth and Bendix 1970; Baader and Nipkow 1998), to mechanise working with string diagrams.

1.1. *Structure of the paper*

In Section 2, we introduce and motivate graphical theories using boolean circuits and tensor networks. We also note key challenges in working with these systems using traditional graph-based methods. After reviewing some of these methods in Section 3, we define selective adhesive functors in Section 4. These give an abstract characterisation for categories that sit inside an ambient adhesive category, and inherit enough properties to support rewriting. We define open-graphs in Section 5 and show that they have a selective adhesive functor into a slice category over **Graph**. In Section 6, we demonstrate how open-graphs can be composed and decomposed, and use these operations for rewriting open-graphs in Section 7. Section 8 defines graphical signatures and shows how they can be used to construct typed open-graphs. Section 9 uses typed open-graphs to construct a monoidal category of cospans and shows how such categories correspond to the free constructions of monoidal categories over a graphical signature. We also show how PROPs can be defined in this language. Finally, we give our conclusions and discuss future work in Section 10.

2. Motivating examples

We begin by introducing two examples to motivate the use of open-graphs for computation. The first is the familiar language of boolean circuits. Boolean circuits are formed by taking basic logic gates and plugging them together. For instance, we can represent the logical expression '$a \wedge (b \wedge \neg c)$' as the graph
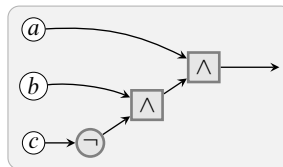

Notice that the output wire of this graph does not end at a vertex. We call this a *half-edge*. We can also represent inputs to a circuit as half-edges. In the above example,

this removes the need to introduce the variables a, b, and c as inputs to the circuit. Instead, we represent the inputs as half-edges:
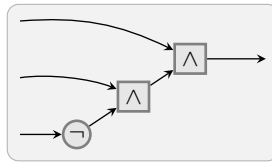

Now suppose we wanted to introduce an expression like '$a \wedge (\neg a \wedge b)$'. We can do this without introducing explicitly named variables by introducing a 'copy' operation:
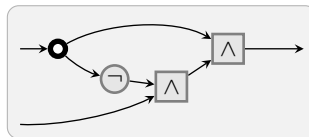
We can also introduce an explicit 'ignore' operation that takes one input and produces no output. To sum up, our language has the following generators, where b is a boolean value:

And   Not   Copy

Boolean value   Ignore
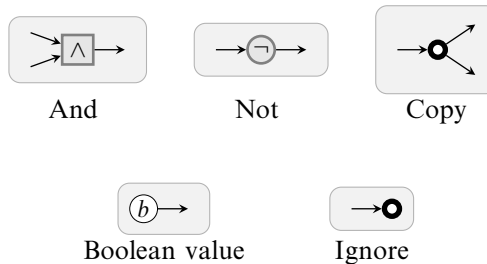
Copies of these components can then be connected together by joining outputs to inputs to form compound circuits. While this is a simple language, it includes satisfiability questions, which are formed by asking whether a given graph can be rewritten to the single boolean value for True. To answer such questions, and more generally to describe the dynamics of boolean circuits, some axioms need to be introduced. For copying and ignoring values, these are
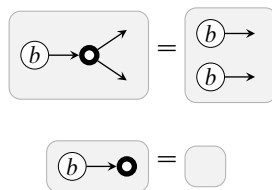
The axioms for conjunction (and-gates: ∧) and negation (not-gates: ¬) are
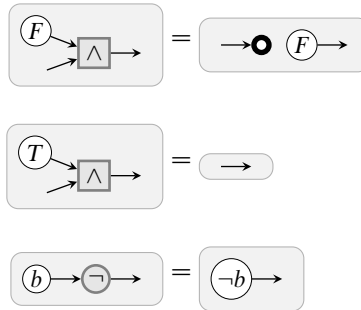
where the boolean values are written as *T* for True and *F* for False. These rules characterise the computational aspects of boolean circuits. The output of a circuit can be evaluated by applying the axioms from left to right. The equations can also be used to simplify circuits.

Although the above rules are sufficient for evaluation (when a circuit has all inputs given), they cannot prove all true equations about boolean circuits. To get a complete set of equations, some additional graphical rules are needed. For instance, the following rule, for double negation elimination, is not directly derivable from those presented earlier:

However, such circuits can be verified by exhaustive analysis directly in the graphical language: we can evaluate every combination of input values to a graphical equation to see if the left- and right-hand sides always evaluate to the same result. This corresponds to a proof by exhaustive case analysis, much like verification by truth-tables.

Once there are sufficient equations, new rules can also be derived directly, without examining all cases. For example, using the double-negation equation above with the evaluation axioms allows the following derivation:
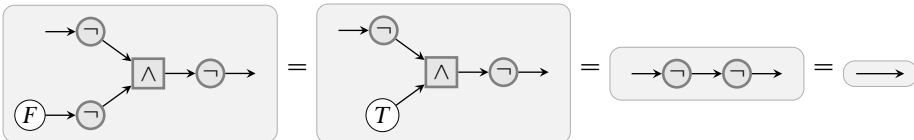
This proves that giving *F* to the compound or-gate is the same as the identity on the other input. This example illustrates how rewriting helps us find identities without resorting to (exponential) case analysis. Moreover, rules in a derivation can be applied simultaneously to separate parts of a graph to parallelise a computation or derivation.

Another salient feature of graph-based representations is that certain aspects of sharing and binding can be described using graphical structure. For example, consider the following rule:
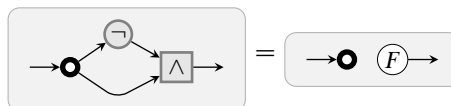
With a formula-based notation, this could be described by an equation between lambda-terms: '$\lambda x.\ ((\neg x) \wedge x) = \lambda x.\ F$'. Graphical notation can treat certain forms of binding by the

structure of edges, with function application of a formula corresponding to composition along half-edges. For example consider applying the left-hand side of the equation to the term F, giving the lambda-term '$\lambda x.\,((\neg x) \wedge x)\,F$'. In this situation, beta-reduction, which reduces the formula to '$(\neg F) \wedge F$', corresponds to an application of the copying rule. In the graphical language, the beta-reduction step is
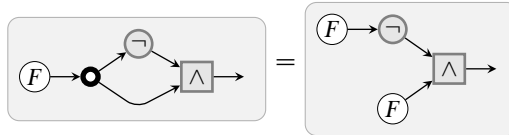
Notice that the graphical representation controls copying carefully: by explicit application of equational rules. This is an essential feature when one moves from *classical* circuits to *quantum* circuits, where copying can only happen in restricted situations (Coecke and Duncan 2008).

We move now from the familiar case of logic circuit rewriting to an example from linear algebra. In (multi-)linear algebra, differential geometry and physics, many computations can be performed using networks of *tensors*. A tensor is a set of real or complex numbers, indexed by one or more integers. For example, the following is an $(n_1 \cdot n_2 \cdot n_3)$-dimensional tensor indexed by 3 integers:

$$\{\chi_{ij}^k : i = 1..n_1;\ j = 1..n_2;\ k = 1..n_3\}$$

Tensors are written with both subscript indices, which serve the purpose of inputs, and superscript indices, which are outputs. Familiar examples of tensors are vectors v^i and matrices M_j^i. We can compose tensors by *contraction*, that is, by 'summing together' a lower index and an upper index of the same dimension:

$$\xi_j^i = \sum_{kl} \chi_{kl}^i \beta_j^k \rho^l.$$

We can simplify such expressions by using the Einstein summation convention, where any repeated indices are assumed to be summed over. However, even with this convention, contraction expressions can get quite complex. Consider this expression, involving six tensors:

$$\alpha_{abc}^{de} \beta_f^{bfg} \gamma_{dh}^i \rho_i^h \phi_{eg}^{jk} \delta_l^l. \tag{1}$$

In order to understand this expression, we have to keep track of 11 indices, which makes computations time-consuming and error-prone. We can instead represent this expression using a graphical language introduced by Penrose (Penrose 1971). Tensors are drawn as boxes, and summations over pairs of indices as wires. The 'identity' tensor (that is, the Dirac delta δ_i^j) is also drawn as a wire. The un-summed, or 'free', indices are left as dangling wires, and sums $\sum \delta_i^i$ are represented as circles. In the graphical notion, expression (1) becomes the following diagram:
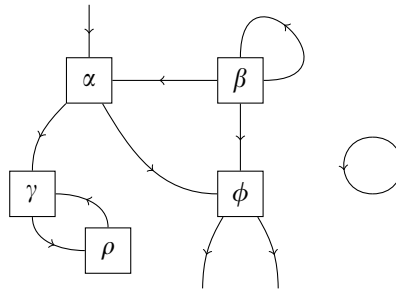
These diagrams are called tensor networks. We can then work directly with these graphs, expressing equations of tensor expressions as graph rewrite rules:
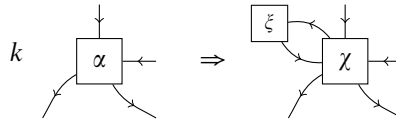
More generally, circuit diagrams, tensor networks, and many other graphical formalisms, can be expressed as arrows in some symmetric monoidal category. The diagrams above can then be interpreted as examples of a diagrammatic language common to all symmetric monoidal categories. These kinds of graphical language introduce a particular challenge to formalising rewriting. For instance, consider a simple graph containing a self loop:

and a rewrite rule that rewrites the box to a line:
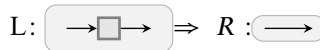
Then, the graph resulting from rewriting the box with a self loop should be a circular edge with no vertices

Graphs with this shape go beyond the normal notion of what one might consider to be a 'graph', yet in many contexts they have a well-behaved interpretation. For instance, in tensor networks, this is the trace of the identity matrix, that is, the dimension of the underlying vector space.

Suppose we tried naively to formalise this situation by representing half-edges as edges connected to 'dummy' points at the boundary:
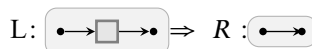
This means the left-hand side of the rewrite does not occur as a subgraph of G. So maybe we could make an exception and not require that L be a subgraph of G, but just have *some* mapping on to G. If we do this, the box and both dummy points could be mapped on to the box in G. However, the result of removing the image of L and replacing it with R is a line, not a circle. A graph that previously had no inputs or outputs is rewritten to a graph with one input and one output, which contradicts the interpretation of rewrite rules as representing some kind of 'local' identity on a diagram. We could make an exception here, but one quickly becomes overwhelmed by the number of special cases that need consideration. We will address this problem uniformly, in Section 5, by allowing *edge-points*. These are extra 'dummy' points that can be introduced not only at the boundaries of graphs, but *along* edges as well. This allows rewrites to be performed in a localised manner, without compromising the validity of the graph as a whole.

3. Related work

There is a significant strand of work related to graph transformations (Ehrig *et al.* 2006; Baldan *et al.* 2008) and rewriting with graph-based presentations of computational processes (Lafont 2010; Lafont and Rannou 2008; Lafont 2003; Lafont 1990). An extension of these formalisms, known as bigraphs, provides another general formalism for graphical rewriting (Milner 2006). Bigraphs are more complex in that they use hypergraphs and introduce a rich hierarchical structure. Another formalism for graphs, called *site-graphs*, is used in systems biology (Danos and Laneve 2004). These give each vertex a set of 'sites' to which edges can be be connected. The distinction between these forms of graphical rewriting and our formalism is that we have an extended notion of a 'graph' that allows for edges to be dangling at one or both ends, or to be connected to themselves. We also consider these graphs as having a fixed interface, drawn as a collection of input and output wires, and only consider graph rewrite rules that preserve this interface. In this regard, we provide a kind of static checking for well-behaved graph transformation systems, much like types do for functional programs. This property is crucial to the graphical formalisms of many of the systems we wish to model. Where our constructions and those of traditional graph transformation share significant similarity is in their reliance on adhesive categories (Lack and Sobocinski 2005) and the *double-pushout* construction for graph rewriting (Ehrig *et al.* 1973). The main difference is that we are working within a different category, namely, that of open-graphs, which enjoy a representation for half-edges. In addition, our construction uses the presentation of typed graphs as a slice over the (adhesive) category of graphs, as presented in Prange *et al.* (2008). In this way, our theory can be viewed as a concrete realisation of the theory of adhesive categories and DPO rewriting, as well as a bridge from this work to the (computational) study of monoidal categories. Part of our contribution can thus be viewed as an elucidation of the conditions for graph rewriting to connect processes along half-edges.

Maps in many monoidal categories admit rich graphical languages (Selinger 2009). These languages become particularly interesting in the study of algebraic structures within monoidal categories. There is a developing field in category theory that investigates these algebras, and how they interact. Lack (2004) showed that a certain class of these monoidal algebras, called *PROPs*, can be composed in much the same way as Beck showed we can compose monads (Appelgate *et al.* 1969). Even richer notions of interacting graphical structures have found applications in the study of non-commuting observables (Coecke and Duncan 2008) and entanglement (Coecke and Kissinger 2010) in quantum mechanics.

In earlier work, we presented a formalism for reasoning about categorical models of quantum information (Dixon and Duncan 2009). In Dixon *et al.* (2010), we proposed several improvements on this early work and suggested that matching and composition became dual notions. In the current paper, we have fleshed out the formalism in the context of adhesive categories, proved the key properties and shown how to construct models of monoidal theories.

4. Selective adhesive functors and rewriting

Adhesive categories provide a useful and quite general setting for performing rewrites on graph-like structures. The distinguishing characteristic of adhesive categories is that pushouts along monomorphisms behave particularly well with respect to pullbacks. The categories we introduce for open-graphs are not exactly adhesive categories, but they live *inside* adhesive categories and inherit 'enough adhesivity' to permit graph rewriting.

In particular, we introduce categories for open-graphs that are subcategories of slices over the category **Graph** of directed graphs and graph homomorphisms. Since a slice over an adhesive category is adhesive (Lack and Sobocinski 2005) and **Graph** is an adhesive category, our categories of open-graphs have inclusions into adhesive categories. To make use of ambient adhesive categories, we define a suitably well-behaved inclusion functor, called a selective adhesive functor. This is well behaved in the sense that essential adhesivity properties for rewriting can be passed back to the subcategory. To define these functors, we first recall the notion of a van Kampen square.

Definition 4.1. A van Kampen square is a pushout

$$
\begin{array}{ccc}
A & \longrightarrow & B \\
\downarrow & & \downarrow \\
C & \longrightarrow & D
\end{array}
$$

such that for any commutative cube

$$
\begin{array}{ccc}
G & \longrightarrow & H \\
& E \longrightarrow F & \\
& & \\
& A \longrightarrow B & \\
C & \longrightarrow & D
\end{array}
$$
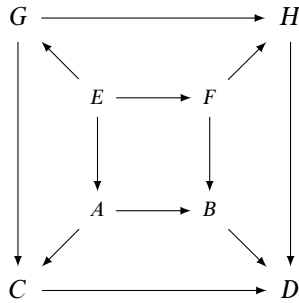
where the back and left faces ($ABEF$ and $ACEG$) are pullbacks, the following are equivalent:

— the front and right faces ($CDGH$ and $BDFH$) are pullbacks;
— the top face ($EFGH$) is a pushout.

Definition 4.2. A category $\mathscr{A}$ is said to be *adhesive* if:

(1) $\mathscr{A}$ has pushouts along monomorphisms;
(2) $\mathscr{A}$ has pullbacks; and
(3) pushouts along monomorphisms in $\mathscr{A}$ are van Kampen squares.

A crucial property of adhesive categories is that certain kinds of *pushout complements* are well defined. These allow us to 'subtract' certain subobjects from an object in a coherent way.

Definition 4.3. A *pushout complement* for a pair of arrows ($m : A \to B, g : B \to D$) is another pair of arrows (f, n) such that

$$
\begin{array}{ccc}
A & \xrightarrow{\;m\;} & B \\
{\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\
C & \xrightarrow{\;n\;} & D
\end{array}
$$
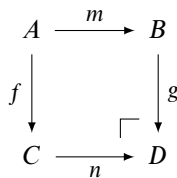
is a pushout.

We first need a few basic lemmas before showing that pushout complements in an adhesive category are unique.

Lemma 4.4 (Lack and Sobocinski 2005). In an adhesive category:

— monomorphisms are stable under pushout; and
— pushouts along monomorphisms are also pullbacks.

Proof. Let m be a monomorphism, and let the following diagram be a pushout:

$$
\begin{array}{ccc}
A & \xrightarrow{\ m\ } & B \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle g} \\
C & \xrightarrow[\ n\ ]{} & D
\end{array}
$$

We need to show that n is mono and the above square is also a pullback. We first construct a commutative cube, containing two copies of the given pushout square: one on the bottom face and one on the right face. We then place a copy of f in the upper-left corner, and fill in the rest with identities:

All of the faces of this cube commute trivially. It follows from this construction that the top face is a pushout, the left face is a pullback and the back face is a pullback if and only if m is a monomorphism. Since m is defined to be a monomorphism and the pushout we started with is a VK-square, adhesivity shows that the front and right faces must be pullbacks. Since the front face is a pullback, n is a monomorphism, so monomorphisms are stable under pushout. Furthermore, since the right face is the pushout we started with, we have also shown that pushouts along monomorphisms are pullbacks. □

We now provide a few lemmas about pullbacks and pushouts that hold in *any* category.

Lemma 4.5. In the following commutative diagram, if the right square is a pullback, the outer square is a pullback if and only if the left square is a pullback:

$$
\begin{array}{ccccc}
A & \xrightarrow{\ f\ } & B & \xrightarrow{\ g\ } & C \\
\downarrow{\scriptstyle h} & & \downarrow{\scriptstyle i} & & \downarrow{\scriptstyle j} \\
D & \xrightarrow[\ k\ ]{} & E & \xrightarrow[\ l\ ]{} & F
\end{array}
$$

Proof. The 'if' direction is trivial. For the other direction, we assume the outer square is a pullback. For an object A', let $f' : A' \to B$ and $h' : A' \to D$ be arrows such that $if' = kh'$:

$$
\begin{array}{ccccc}
A & \xrightarrow{\;f\;} & B & \xrightarrow{\;g\;} & C \\
\downarrow{\scriptstyle h} & & \downarrow{\scriptstyle i} & & \downarrow{\scriptstyle j} \\
D & \xrightarrow{\;k\;} & E & \xrightarrow{\;l\;} & F
\end{array}
$$

with A' mapping via f' to B and via h' to D.
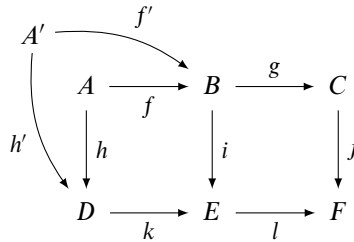
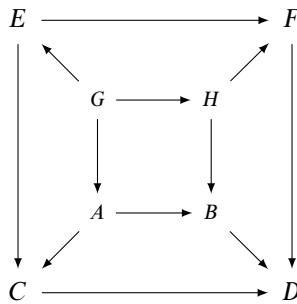Then gf' and h' form a cone under the outer pullback, so there exists a unique $u : A' \to A$ such that $gf' = gfu$ and $h' = hu$. From the universal property of the right pullback, it follows that $f' = fu$, so the left square is a pullback. $\qquad\square$

Lemma 4.6. For a commutative cube, where the front, right and back faces are pullbacks, the left face is also a pullback:

$$
\begin{array}{ccc}
E & \longrightarrow & F \\
& G \longrightarrow H & \\
& A \longrightarrow B & \\
C & \longrightarrow & D
\end{array}
$$

.

Proof. As the back and right faces are both pullback squares, we can apply Lemma 4.5 to show that the back and right faces taken together form a larger pullback square. Then, by commutativity of the cube, the square formed by the left and front faces taken together is also a pullback square. Applying Lemma 4.5 from right to left shows that the left face is a pullback. $\qquad\square$

Lemma 4.7. Isomorphisms are stable under pushout. That is, for the pushout

$$
\begin{array}{ccc}
A & \xrightarrow{\;\phi\;} & B \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle p} \\
C & \xrightarrow{\;q\;} & D
\end{array}
$$
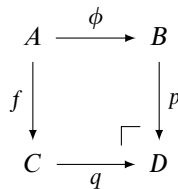
if ϕ is an isomorphism, so too is q.

Proof. Let $f\phi^{-1}$ and 1_C form a cocone to C over the pushout. It follows straightforwardly that the induced map $q' : D \to C$ is the inverse of q. □

We are now ready to prove the uniqueness theorem for pushout complements.

Theorem 4.8 (Lack and Sobocinski 2005). *If a pair of arrows* (m, g), *where* m *is mono, has a pushout complement, it is unique up to isomorphism. That is, for any two pushout complements,* (f, n) *and* (f', n'), *there exists an isomorphism* ϕ *making the following diagram commute:*

$$\begin{array}{ccc} A & \xrightarrow{f} & C \\ f' \downarrow & \overset{\phi}{\diagdown} & \downarrow n \\ C' & \xrightarrow{n'} & D \end{array} \qquad (2)$$

Proof. Suppose both of the following are pushout squares:

$$\begin{array}{ccc} A & \xrightarrow{m} & B \\ f \downarrow & & \downarrow g \\ C & \xrightarrow{n} & D \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{m} & B \\ f' \downarrow & & \downarrow g \\ C' & \xrightarrow{n'} & D \end{array}$$

Following a similar strategy to Lemma 4.4, we use these squares to build a commutative cube

where the first pushout square forms the bottom face, and the second pushout square forms the right face. The front face is the pullback of n and n', and the back face is the pullback of m with itself. This pullback consists of identities because m is mono. Now, f and f' form a cone under the pullback of n and n', so we let h be the induced map.

By Lemma 4.4, the right face is a pullback. We can therefore conclude from Lemma 4.6 that the left face is also a pullback. From the VK-square property, we can then conclude that the top face is a pushout. By Lemma 4.7, k is an isomorphism.

We can also form a similar cube, but with the positions of the two pushouts interchanged:
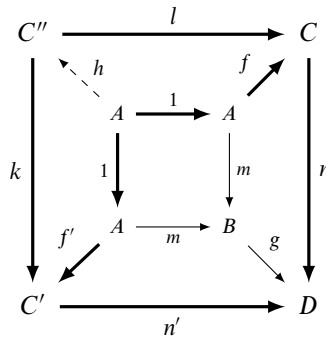
We can then conclude similarly that l is an isomorphism. By construction, all faces commute, so by letting $\phi := kl^{-1}$, we can read off the statement of the theorem from the commutative cube (with the relevant arrows shown in bold). □

In order to define subcategories of adhesive categories, where a selected class of pushout squares has unique pushout complements, we define a selective adhesive functor.

Definition 4.9 (selective adhesive functor). Let $\mathscr{C}$ be a category and $\mathscr{A}$ be an adhesive category. A functor $S : \mathscr{C} \to \mathscr{A}$ is said to be a *selective adhesive functor* if it:

(1) is faithful;
(2) preserves monomorphisms;
(3) creates isomorphisms; and
(4) reflects pushouts.

Recall that a functor $S : \mathscr{C} \to \mathscr{A}$ *creates isomorphisms* in the sense of Adamek *et al.* (2009) if, for any isomorphism $\phi : X \to SY$ in $\mathscr{A}$, there exists a unique object X' and arrow $\phi' : X' \to Y$ such that $S(X') = X$ and $S(\phi') = \phi$, and, furthermore, ϕ' is an isomorphism in $\mathscr{C}$.

For our purposes, the most important example of a selective adhesive functor is the embedding of the category of open-graphs (respectively, T-open-graphs) into the adhesive category of $\mathscr{G}_2$-typed graphs (respectively, $\mathscr{G}_T$-typed graphs). The definition of this category and concrete examples of many of the remaining constructions in this section can be found in Section 5. These two embeddings are shown to be selective adhesive functors in Theorems 5.10 and 8.12.

We now continue with a presentation of the general properties of selective adhesive functors for rewriting.

Definition 4.10 (S-adhesive spans and pushouts). Let $S : \mathscr{C} \to \mathscr{A}$ be a selective adhesive functor. A span $A \xleftarrow{f} B \xrightarrow{g} C$ in $\mathscr{C}$ is said to be an *S-adhesive span* if it has a pushout, and that pushout is preserved by S. Such pushouts are called *S-adhesive pushouts*.

Since S reflects *all* pushouts, we could also define S-adhesive spans as spans that have a pushout reflected by S.

Definition 4.11 (S-adhesive pushout complement). An S-*adhesive pushout complement* for a pair of arrows (b, f) is a pushout complement where the following diagram is an S-adhesive pushout:

$$
\begin{array}{ccc}
B & \xrightarrow{\ b\ } & K \\
{\scriptstyle c}\downarrow & & \downarrow{\scriptstyle f} \\
G' & \xrightarrow[\ g\ ]{} & G
\end{array}
$$

The map b is called the *boundary* of K and c is called the *coboundary* of K in G.

Informally, G' should be thought of as G with K cut out from it, where b identifies the boundary of K, and the coboundary c identifies the boundary of where K was cut out from G.

When it is convenient, we shall use the subtraction notation $G -_{b,f} K := G'$ to denote the pushout complement defined above. In later sections, the boundary map b will be uniquely defined by K, so we shall then simply write $G -_f K$. Since the categories we are concerned with come with a canonical notion of boundary, we typically only require that the boundary of K be mono; unlike Prange *et al.* (2008), which requires the induced pushout to satisfy an initiality condition.

Lemma 4.12. If a pair of arrows (b, f), where b is mono, have an S-adhesive pushout complement, it is unique up to isomorphism.

Proof. Let (c, g) and (c', g') be S-adhesive pushout complements. Then the following diagrams are pushouts in the adhesive category $\mathscr{A}$:

$$
\begin{array}{ccc}
SB & \xrightarrow{\ Sb\ } & SK \\
{\scriptstyle Sc}\downarrow & & \downarrow{\scriptstyle Sf} \\
SG' & \xrightarrow[\ Sg\ ]{} & SG
\end{array}
\qquad\qquad
\begin{array}{ccc}
SB & \xrightarrow{\ Sb\ } & SK \\
{\scriptstyle Sc'}\downarrow & & \downarrow{\scriptstyle Sf} \\
SG'' & \xrightarrow[\ Sg'\ ]{} & SG
\end{array}
$$

Since S preserves monos, these are both pushout complements of (Sb, Sf) for Sb mono. So this diagram commutes in $\mathscr{A}$, for ϕ' an isomorphism:

$$
\begin{array}{ccc}
SB & \xrightarrow{\ Sc\ } & SG' \\
{\scriptstyle Sc'}\downarrow & \overset{\phi'}{\diagdown} & \downarrow{\scriptstyle Sg} \\
SG'' & \xrightarrow[\ Sg'\ ]{} & SG
\end{array}
$$

Since S creates isomorphisms, there exists an iso $\phi : G' \to G''$ such that $S\phi = \phi'$. Substituting this map into the previous diagram, we get

$$
\begin{array}{ccc}
SB & \xrightarrow{Sc} & SG' \\
{\scriptstyle Sc'}\Big\downarrow & {\scriptstyle S\phi} \diagdown & \Big\downarrow{\scriptstyle Sg} \\
SG'' & \xrightarrow[Sg']{} & SG
\end{array}
$$

Diagram (2) commutes by the faithfulness of S. $\qquad\qquad\square$

Definition 4.13 (rewrite rule). A rewrite rule $L \multimap_{b_1,b_2} R$ is a span of monomorphisms:

$$
L \xleftarrow{b_1} B \xrightarrow{b_2} R.
$$

For brevity, we will often denote a rewrite rule simply as $L \multimap R$, leaving the boundary maps implicit. However, when we do this, each time we write $L \multimap R$, it will denote the same rewrite rule, and, in particular, it will have the same boundary maps.

Definition 4.14 (S-matching). For a rewrite rule $L \multimap_{b_1,b_2} R$, a monomorphism $m : L \to G$ is called an *S-matching* if $B \xrightarrow{b_1} L \xrightarrow{m} G$ has an S-adhesive pushout complement.

Definition 4.15 (S-adhesive rewrite). Let $L \multimap_{b_1,b_2} R$ be a rewrite rule and $m : L \to G$ be an S-adhesive matching. Then for G' the S-adhesive pushout complement of $B \xrightarrow{b_1} L \xrightarrow{m} G$, the following diagram is called an *S-adhesive rewrite* if the right-hand pushout is S-adhesive:
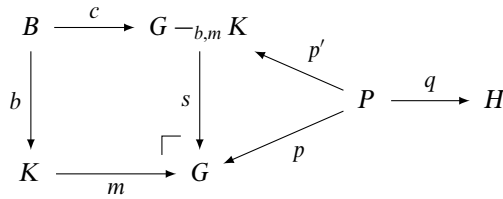
$$
\begin{array}{ccccc}
L & \xleftarrow{b_1} & B & \xrightarrow{b_2} & R \\
{\scriptstyle m}\Big\downarrow & & \Big\downarrow{\scriptstyle c} & & \Big\downarrow \\
G & \longleftarrow & G' & \longrightarrow & H
\end{array}
$$

In such a case, we write H as $G[L \multimap_{b_1,b_2} R]_m$.

Note that the left-hand pushout above is also S-adhesive by the definition of S-matching. We often do not care about the particular rewrite rule and matching used to rewrite one graph into another, but merely that there *exists* such a rewrite involving a rule in some fixed set. For this, we introduce rewrite systems and a 'rewrites-to' relation.

Definition 4.16 (rewrite system). A set of rewrite rules $\mathbb{S}$ is called a *rewrite system*. We define the relation $G \rightarrowtriangle_{\mathbb{S}} H$ to mean there exists a rule $L \multimap R \in \mathbb{S}$ and an S-adhesive matching $m : L \to G$ such that $H \cong G[L \multimap R]_m$. The reflexive, transitive closure of $\rightarrowtriangle_{\mathbb{S}}$ is denoted $\overset{*}{\rightarrowtriangle}_{\mathbb{S}}$, and the reflexive, symmetric, transitive closure as $\overset{*}{\leftrightarrowtriangle}_{\mathbb{S}}$.

Theorem 4.17. *S*-adhesive pushout complements commute with *S*-adhesive pushouts. Consider the following commuting diagram, where b is mono, (b, m) has an *S*-adhesive pushout complement, and (p, q) and (p', q) are both *S*-adhesive spans:
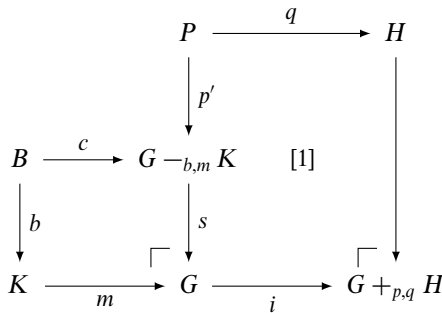
$$
\begin{array}{ccc}
B & \xrightarrow{\ c\ } & G -_{b,m} K \\
\downarrow{\scriptstyle b} & & \downarrow{\scriptstyle s} \\
K & \xrightarrow{\ m\ } & G
\end{array}
\qquad
\begin{array}{c}
P \xrightarrow{\ q\ } H
\end{array}
$$

(with p' from P to $G -_{b,m} K$ and p from P to G)

Then, for the pushout injections

$$i : G \hookrightarrow G +_{p,q} H$$
$$i' : G -_{b,m} K \hookrightarrow (G -_{b,m} K) +_{p',q} H,$$

there is an open-graph isomorphism commuting with the coboundaries c and c' of K in G and $G +_{p,q} H$, respectively:

$$
\begin{array}{ccc}
B & \xrightarrow{\ c'\ } & (G +_{p,q} H) -_{b,im} K \\
\downarrow{\scriptstyle c} & & \downarrow{\scriptstyle \cong} \\
G -_{b,m} K & \xrightarrow{\ i'\ } & (G -_{b,m} K) +_{p',q} H
\end{array}
\qquad (3)
$$

Proof. The proof follows from the associativity of pushouts and the uniqueness of pushout complements. First note that in the following diagram, [1] commutes and is a pushout because $sp' = p$:

$$
\begin{array}{ccccc}
 & & P & \xrightarrow{\ q\ } & H \\
 & & \downarrow{\scriptstyle p'} & & \\
B & \xrightarrow{\ c\ } & G -_{b,m} K & [1] & \\
\downarrow{\scriptstyle b} & & \downarrow{\scriptstyle s} & & \downarrow \\
K & \xrightarrow{\ m\ } & G & \xrightarrow{\ i\ } & G +_{p,q} H
\end{array}
$$

By the associativity of pushouts, the following diagram also commutes, and the marked squares are pushouts:

$$
\begin{array}{ccc}
P & \xrightarrow{\quad q \quad} & H \\
\downarrow{\scriptstyle p'} & & \downarrow \\
B \xrightarrow{\ c\ } G -_{b,m} K & \longrightarrow & (G -_{b,m} K) +_{p',q} H \\
\downarrow{\scriptstyle b} \quad\quad [2] & & \downarrow \\
K \xrightarrow{\ m\ } G & \xrightarrow{\ i\ } & G +_{p,q} H
\end{array}
$$

Now compare [2] with the subtraction of $im : K \to G +_{p,q} H$:

$$
\begin{array}{ccc}
B & \longrightarrow & (G +_{p,q} H) -_{b,im} K \\
\downarrow{\scriptstyle b} & & \downarrow \\
K & \xrightarrow{\ im\ } & G +_{p,q} H
\end{array}
$$

The result then follows from the uniqueness of pushout complements. $\square$

Theorem 4.18. *S*-adhesive rewrites commute with *S*-adhesive pushouts. Let $m : L \to G$ be an *S*-matching of $L \multimap_{b_1, b_2} R$. The rewrite is computed as the double pushout:

$$
\begin{array}{ccccc}
L & \xleftarrow{\ b_1\ } & B & \xrightarrow{\ b_2\ } & R \\
\downarrow{\scriptstyle m} & & \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle m'} \\
G & \xleftarrow{\ s\ } & G -_{b,m} L & \xrightarrow{\ s'\ } & G[L \multimap R]_m
\end{array}
$$

Let (p, q), (p', q) and $(\widehat{p}, q)$ be three adhesive spans such that

$$
\begin{array}{c}
G \\
\uparrow{\scriptstyle s} \\
G -_{b_1, m} L \xleftarrow{\ p'\ } P \xrightarrow{\ q\ } H \\
\downarrow{\scriptstyle s'} \\
G[L \multimap R]_m
\end{array}
$$

with curved arrows p (from P to G) and $\widehat{p}$ (from P to $G[L \multimap R]_m$) \hfill (4)

Then, for the pushout injection $i : G \to G +_{p,q} H$, if im is mono, the following is an isomorphism:

$$(G[L \multimap R]_m) +_{\hat{p},q} H \cong (G +_{p,q} H)[L \multimap R]_{im}.$$

Proof. Since pushout complements are unique up to isomorphism, we can choose $(G -_{b_1,m} L)$ to be equal to $((G[L \multimap R]_m) -_{b_2,m'} R)$, for the same coboundary c. Then, by two applications of Theorem 4.17, we can choose

$$(G -_{b_1,m} L) +_{p',q} H = ((G[L \multimap R]_m) -_{b_2,m'} R) +_{p',q} H$$

as the pushout complement of both of the following squares:

$$
\begin{array}{ccccc}
L & \longleftarrow & B & \longrightarrow & R \\
\downarrow^{im} & & \downarrow^{c'} & & \downarrow \\
G +_{p,q} H & \longleftarrow & (G -_{b_1,m} L) +_{p',q} H & \longrightarrow & G[L \multimap R] +_{\hat{p},q} H
\end{array}
$$

Note that c' becomes the coboundary for both squares because diagram (3) commutes. This is then exactly the computation of the rewrite $(G +_{p,q} H)[L \multimap R]_{im}$. $\square$

We shall use these two theorems throughout the paper to show that rewriting is compatible with several notions of composing graphs.

5. Open-graphs

In this section we provide a formal definition for the notion of graphs that can contain edges with unconnected ends, called *open-graphs*. We do this by introducing a special kind of graph with two distinct types of points. It has points that should be considered as 'real' vertices, and other intermediate points, called edge-points, that occur along edges. In this construction, the 'logical' edges of an open-graph, which we also call *wires*, can be presented as chains of edge-points, which need not have a vertex at either end. Thus we can define the boundary of an open-graph as the unconnected ends of these wires. This provides the interface through which we connect open-graphs together.

We define the category **OGraph** as a full subcategory of the (adhesive) category of $\mathscr{G}_2$-typed graphs, **Graph**/$\mathscr{G}_2$. We then show that the inclusion $S :$ **OGraph** $\hookrightarrow$ **Graph**/$\mathscr{G}_2$ is a selective adhesive functor.

We begin with the standard definition of the category **Graph** of directed graphs in order to fix some notation.

Definition 5.1. Let **Graph** be the category of graphs. It is defined as the functor category $[\mathbb{G}, \mathbf{Set}]$ for $\mathbb{G}$ defined as

$$E \underset{t}{\overset{s}{\rightrightarrows}} P .$$

E identifies the edges of the graph, and P the points. s and t are functions taking an edge to its source and target respectively. If $t(e) = p$, then e is called an *in-edge* of p, and if $s(e) = p$, then e is called an *out-edge* of p.

Note that our language for graphs differs slightly from the convention in that we use the term 'point' rather than 'vertex'. The reason for this will become clear once we introduce a typing on points. The typegraph $\mathscr{G}_2$ will be used to distinguish points that should be interpreted as 'logical' vertices from the 'dummy'-points that occur along an edge:

$$\mathscr{G}_2 := \overset{\frown}{(V)} \rightleftarrows \overset{\frown}{(\epsilon)} \overset{\frown}{\leftarrow}$$

A graph G is said to be typed by $\mathscr{G}_2$ when there is a typing morphism $\tau : G \rightarrow \mathscr{G}_2$. When a point $p \in G$ is mapped to V, that is, $\tau(p) = V$, we refer to it as a *vertex*. The other points in G, those with $\tau(p) = \epsilon$, are called *edge-points*.

Definition 5.2 (OGraph). The category **OGraph** of *open-graphs* is the full subcategory of the slice category **Graph**/$\mathscr{G}_2$. Objects are those of **Graph**/$\mathscr{G}_2$ where each edge-point has at most one in-edge and one out-edge.

This slice construction allows open-graphs to be represented as graphs with a typing morphism to $\mathscr{G}_2$. The slice construction also plays two further roles: it distinguishes 'real' vertices from edge-points, and the lack of a self-loop on V ensures that every path between two vertices must have at least one edge-point.

Example 5.3. The following is a diagrammatic presentation of an open-graph:
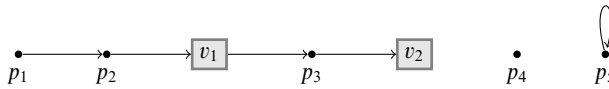
This diagram abbreviates an open-graph with its morphism to $\mathscr{G}_2$, which can otherwise be drawn more verbosely as
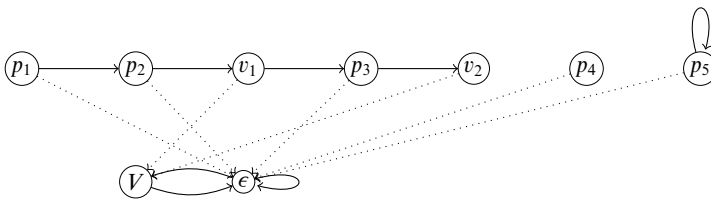
where the dotted arrows indicate the type-morphism for points, and the edge mapping is inferred trivially.

We now show some basic properties of **OGraph**. In particular, we develop the properties needed to show that the inclusion of **OGraph** into **Graph**/$\mathscr{G}_2$ is a selective adhesive functor.

**Lemma 5.4.** Any subgraph of an open-graph is also an open-graph.

Proof. Suppose G is an open-graph and H is a subgraph of G. Then every edge-point in H must have at most one in-edge and at most one out-edge. □

Lemma 5.5. A map in **OGraph** is a monomorphism if and only if it is injective.

Proof. The fact that injective maps are monomorphism follows from morphisms in **Graph**/$\mathscr{G}_2$ being monos and the faithfulness of the inclusion functor from **OGraph** into **Graph**/$\mathscr{G}_2$.

We now prove that if a morphism in **OGraph** is not injective, it is not a monomorphism. A non-injective morphism $f : G \to H$ either maps two or more edges e_i to a single edge, or two or more points p_i to a single point. The smallest sub-open-graph K of H containing $f(e_i)$ or $f(p_i)$ is either a single point or an edge with its end points. We can map the open-graph K onto its pre-image in G using a distinct map g_i for each e_i or p_i in such a way that $f \circ g_i = f \circ g_j$ for all i, j. Thus, there are two or more open-graph morphisms g_i such that the $f \circ g_i$ are equal, and hence f is not mono. $\square$

We will now prove a similar result for surjections. However, since **OGraph** restricts which typed graphs are allowed, not all epimorphisms are surjective. For example, consider
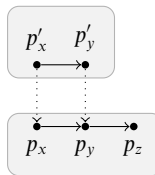
The map indicated by the dotted arrows suffices to distinguish maps from the bottom open-graph under post-composition because fixing the image of p_y uniquely fixes the image of p_z and the edge from p_y to p_z. So it is an epimorphism, but it is clearly not surjective. However, all *strong* epimorphisms are surjective. To show this, we first recall the notion of a strong epimorphism, and prove a simple fact about surjections.

Definition 5.6. A *strong epimorphism* in $\mathscr{C}$ is an epimorphism e that is left-orthogonal to all monomorphisms in $\mathscr{C}$. That is, for any commutative square of the form

$$
\begin{array}{ccc}
A & \xrightarrow{\ e\ } & B \\
f \downarrow & \nearrow^{\ d} & \downarrow g \\
C & \hookrightarrow_{\ m\ } & D
\end{array}
$$
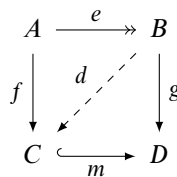
with m as a monomorphism, there exists a unique diagonal map d making the diagram commute.

Lemma 5.7. A map in **OGraph** is a strong epimorphism if and only if it is surjective.

Proof. The fact that surjections in **OGraph** are strong epimorphisms follows from surjections being strong epimorphisms in **Graph**/$\mathscr{G}_2$ and the faithfulness of the inclusion functor.

For the other direction, assume $e : A \to B$ is a strong epimorphism. Let $e' : A \to e[A]$ be the restriction of e to its image (which is also an open-graph by Lemma 5.4), and let

$i : e[A] \to B$ be the inclusion of the image of e in B. Hence, i is a mono, so there exists a map d making the following diagram commute:

$$
\begin{array}{ccc}
A & \xrightarrow{\ e\ } & B \\
{\scriptstyle e'}\downarrow & {}^{d}\nearrow & \downarrow{\scriptstyle 1_B} \\
e[A] & \xhookrightarrow{\ i\ } & B
\end{array}
$$

The inclusion i splits ($i \circ d = 1_B$), so it must be surjective, and thus e is also surjective. □

Lemma 5.8. The embedding functor $S : \mathbf{OGraph} \hookrightarrow \mathbf{Graph}/\mathscr{G}_2$ preserves and reflects monomorphisms and strong epimorphisms.

Proof. This follows from S being an inclusion functor and Lemmas 5.5 and 5.7, and by recalling that in $\mathbf{Graph}/\mathscr{G}_2$, epimorphisms are exactly the surjections and monomorphisms are exactly the injections. □

Lemma 5.9. $\mathbf{OGraph}$ has unique strong epi-mono factorisations.

Proof. Any map f factors through its image, which is an open-graph by Lemma 5.4:

$$
A \xtwoheadrightarrow{\ f_e\ } f[A] \xhookrightarrow{\ f_m\ } B
$$

We now show that this factorisation is unique. Lemmas 5.5 and 5.7 (and the fullness of S) let us conclude that if the factorisation is unique in $\mathbf{Graph}/\mathscr{G}_2$, where epimorphisms are surjections and monomorphisms are injections, then it is unique in $\mathbf{OGraph}$. $\mathbf{Graph}/\mathscr{G}_2$ is a regular category, and all regular categories have unique regular epi-mono factorisations, so $\mathbf{OGraph}$ must have unique strong epi-mono factorisations. □

Theorem 5.10. The embedding functor $S : \mathbf{OGraph} \hookrightarrow \mathbf{Graph}/\mathscr{G}_2$ is a selective adhesive functor.

Proof. S is faithful by definition and preserves monomorphisms by Lemma 5.8. It reflects pushouts because full and faithful functors reflect all colimits. S creates isomorphisms because S is full, faithful and closed under isomorphisms, and every such functor creates isomorphisms. □

Corollary 5.11. The category $\mathbf{OGraph}$ has coproducts given by the disjoint union of the underlying typed graphs.

Proof. The disjoint union does not affect the incidence of individual edge-points, so the disjoint union of two open-graphs is again an open-graph. Since this is the coproduct in $\mathbf{Graph}/\mathscr{G}_2$ and the embedding functor S reflects colimits, it is a coproduct in $\mathbf{OGraph}$. □
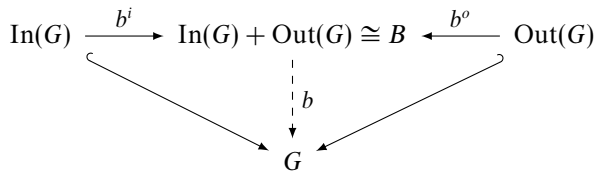
5.1. *Boundaries of open-graphs*

The main goal of defining open-graphs is to have a notion of edges that are not connected to vertices on one or both ends. These serve as inputs and outputs along which open-graphs can be composed. Before we consider the composition of open-graphs, we shall make several convenient definitions relating the boundary of open-graphs, as defined by their inputs and outputs.

Definition 5.12 (OGraph notation). If an edge-point has no in-edges, it is called an *input*. We write the set of inputs of an open-graph G as $\text{In}(G)$. Similarly, an edge-point with no out-edges is called an *output*, and the set of outputs is written $\text{Out}(G)$. The inputs and outputs define an open-graph's *boundary*. If a boundary point has no in-edges and no out-edges (it is both and input and output), it is called an *isolated point*. An open-graph consisting only of isolated points is called a *point-graph*.

Note that when there is no ambiguity, we shall use $\text{In}(G)$ and $\text{Out}(G)$ to also refer to the point-graph containing only the inputs or outputs of G. Considered as open-graphs, these have natural inclusions into G. We will now define the *boundary graph of an open-graph*, which plays a particularly important role for both the composition and decomposition of open-graphs.

Definition 5.13 (boundary graph and boundary map). Given an open-graph G, its *boundary graph* is the point-graph formed from the coproduct of its inputs and outputs: $B := \text{In}(G) + \text{Out}(G)$. The *boundary map* of G is the induced map $b : B \to G$ of the inclusions of $\text{In}(G)$ and $\text{Out}(G)$:

$$\text{In}(G) \xrightarrow{\;b^i\;} \text{In}(G) + \text{Out}(G) \cong B \xleftarrow{\;b^o\;} \text{Out}(G)$$

$$\downarrow b$$

$$G$$

Note that for a boundary map b, we refer to the associated coproduct injections as b^i and b^o.

**Example 5.14.** The folowing diagram shows an open-graph with its boundary graph drawn above it and a boundary map indicated by dotted arrows:
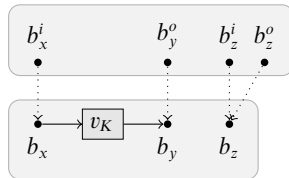
A boundary map identifies all the inputs and outputs of G, and is injective except on isolated points in G, where it is 2-to-1. Because each isolated-point is both an input and an output, a boundary graph has two points for each isolated point in the image

of the boundary map. Note that a boundary map is mono if and only if its codomain open-graph has no isolated points.

Definition 5.15 (sharing the same boundary). Two open-graphs L and R are said to *share the same boundary graph* B by boundary maps b_1 and b_2 when $L \xleftarrow{b_1} B \xrightarrow{b_2} R$, $\text{In}(L) \cong \text{In}(R)$, $\text{Out}(L) \cong \text{Out}(R)$ and the following diagram commutes:
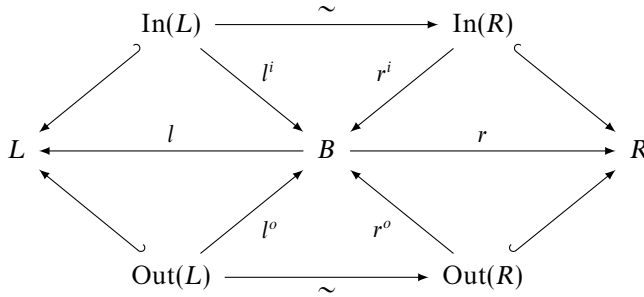
Note that when L and R have no isolated points, this condition means that the boundary graphs are in bijection. Furthermore, this bijection sends inputs to inputs and outputs to outputs.

6. Matching, composition and decomposition for open-graphs

In this section we introduce the notion of *matching*, which defines how one open-graph can be meaningfully identified within another. We then show how open-graphs can be composed and decomposed using the embedding functor $S : \mathbf{OGraph} \to \mathbf{Graph}/\mathscr{G}_2$. The fact that S is a selective adhesive functor then lets us introduce definitions for subtracting one open-graph from another (by pushout complements), and for connecting open-graphs along their boundary.

To define matching, we first introduce the notion of the *edge neighbourhood* of a vertex, and of a morphism being a *local isomorphism*.

Definition 6.1. The *edge neighbourhood* of a vertex v is the set of all edges that are connected to v:

$$N(v) := \{e : s(e) = v \text{ or } t(e) = v\}.$$

Definition 6.2. An open-graph morphism $f : G \to H$ is said to be a *local isomorphism* when, for every vertex $v \in G$, the edge function of f restricts to a bijection on the edge neighbourhood of v:

$$f^v : N(v) \xrightarrow{\sim} N(f(v)).$$

Note that local isomorphism only applies to vertices and not to edge-points.

Definition 6.3 (matching). A monomorphism $m : G \to H$ of open-graphs is said to be a *matching* when it is a local isomorphism. In this case, G is said to *match* H at m.

Matching characterises how one open-graph can be identified as part of a larger one. In particular, the image of a matching has the same 'type' as the matching open-graph in the sense that it has the same number of inputs and outputs. We will refine this notion of typing further in Section 9. The key feature of matching is that it supports meaningful ways to compose and decompose open-graphs. Before defining these operations, we will first define the spans that preserve open-graphs under pushout. The crucial features of such pushouts are:

— If two edges are identified, the full path of edge-points of at least one wire is identified with (part of) the other wire: identification of edges never results in wires that branch.
— The output of one edge is never connected to the output of another edge, and likewise with inputs.

This idea is formalised by the notion of *boundary-coherent spans*.

Definition 6.4 (boundary coherent spans). A span $H_1 \xleftarrow{f} G \xrightarrow{g} H_2$ is said to be *boundary coherent* when f and g are matchings and:

(1) For all $p \in \text{In}(G)$, at least one of $f(p)$ and $g(p)$ is an input.
(2) For all $p \in \text{Out}(G)$, at least one of $f(p)$ and $g(p)$ is an output.

A parallel pair of arrows $f, g : G \to H$ is said to be a *boundary coherent pair* when the span $H \xleftarrow{f} G \xrightarrow{g} H$ is boundary coherent.

Theorem 6.5. Boundary-coherent spans are S-adhesive spans.

Proof. Let $A \xleftarrow{f} B \xrightarrow{g} C$ be a boundary coherent span. Then the following diagram is a pushout in **Graph**$/\mathscr{G}_2$:

$$
\begin{array}{ccc}
SA & \xrightarrow{Sf} & SB \\
{\scriptstyle Sg}\downarrow & & \downarrow{\scriptstyle i_1} \\
SC & \xrightarrow[i_2]{} & D
\end{array}
$$

Since S reflects pushouts, it suffices to show that D, i_1 and i_2 are in the image of S. Since S preserves monos, Sf and Sg are mono. A pushout of monos in **Graph**$/\mathscr{G}_2$ is (up to isomorphism) just a union. So, without loss of generality, we can let $SA = SB \cap SC$ and $D = SB \cup SC$, and the two inclusions of the intersection form a boundary-coherent span. Thus we can rewrite the above pushout as follows:

$$
\begin{array}{ccc}
SB \cap SC & \xhookrightarrow{Sf} & SB \\
{\scriptstyle Sg}\uparrow & & \downarrow{\scriptstyle i_1} \\
SC & \xhookrightarrow[i_2]{} & SB \cup SC
\end{array}
$$

We will now show that $SB \cup SC$ is an open-graph, namely, each edge-point has at most one in-edge and one out-edge. The proof is by contradiction. Suppose an edge-point p in $SB \cup SC$ has two out-edges. Then one must be in SB and the other in SC, so neither are in the intersection, and thus p is an output in $SB \cap SC$. But it is not an output in SB or SC, thus contradicting the boundary coherence assumption. A contradiction follows similarly when p has two in-edges, so $SB \cup SC$ is an open-graph. $\qquad\square$

This provides boundary-coherent spans with nice properties for pushouts as reflected by the selective adhesive functor S. These pushouts, which we call *mergings*, will play a central role in our rewriting construction.

Definition 6.6 (merging). Given a boundary-coherent span $G_1 \xleftarrow{m_1} K \xrightarrow{m_2} G_2$, we use the notation $M := G_1 +_{m_1,m_2} G_2$ for the pushout of the span, which we call the *merging* of G_1 and G_2 on K by m_1 and m_2:
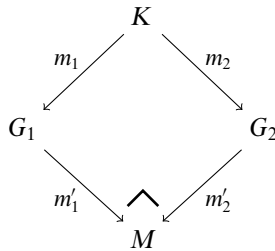
This makes M the smallest open-graph containing G_1 and G_2 with a single copy of the shared subgraph K, as identified by m_1 and m_2.

**Example 6.7.** The following diagram is an illustration of the merging of open-graphs:
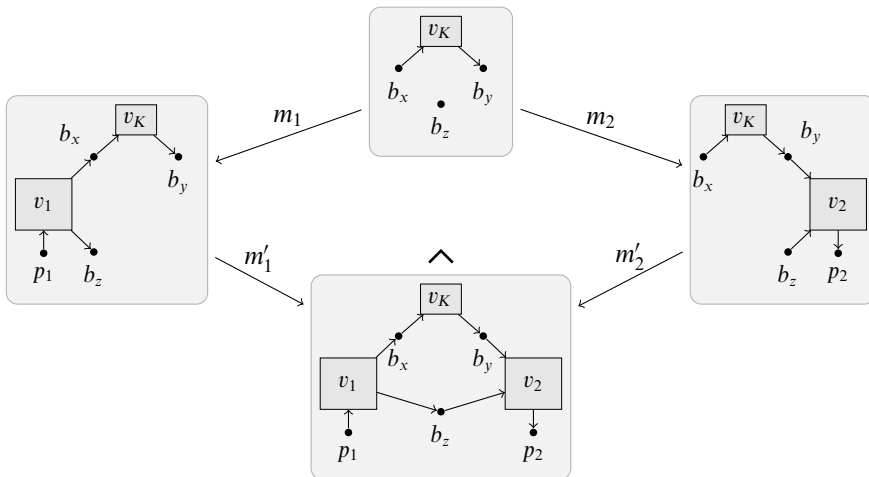
The grey boxes are drawn around the open-graphs involved to distinguish between edges in the open-graphs and the maps of the pushout diagram. The image of the maps are indicated by the naming of edge-points and vertices.

A particularly important special case of merging is composition along half-edges, which we call *plugging*.

Definition 6.8 (plugging). An open-graph merging $G_1 +_{b_1,b_2} G_2$ is called a *plugging* and written $G_1 +^*_{b_1,b_2} G_2$ when the open-graph being merged on is a point-graph: that is, in the span $G_1 \xleftarrow{b_1} P \xrightarrow{b_2} G_2$, the open-graph P is a point-graph.

Example 6.9. The following diagram is an illustration of plugging using pushouts:
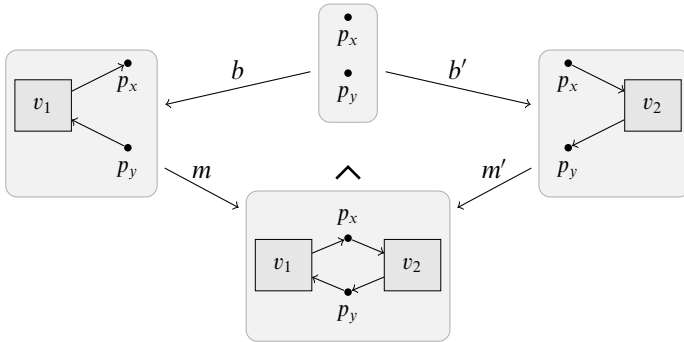
Note that the special case of plugging formed by pushouts on the empty open-graph is the disjoint union of open-graphs, written simply as $G + H$; visually this corresponds to placing open-graphs side-by-side.

Merging and plugging can be understood as special cases of pushouts of graphs. The key additional characteristic being that these pushouts ensure that the 'logical'-edges do not branch. We will use these pushouts in the following section to define rewriting for open-graphs, essentially following a double pushout approach to graph rewriting (Ehrig *et al.* 2006). But we first introduce a dual notion to merging, called *subtraction*, which is formed by S-adhesive pushout complements of matches. Intuitively, subtraction removes part of an open-graph identified by a matching. We will first give a concrete definition for subtraction, and then show that this definition does indeed produce S-adhesive pushout complements.

Definition 6.10 (subtraction). We define the *subtraction* of G from M at a match $m : G \to M$, written $M -_m G$, as the open-graph H defined by

$$P_H = (P_M \backslash m[P_G]) + P_B$$
$$E_H = (E_M \backslash m[E_G])$$
$$s_H(e) = \begin{cases} b^o(p) & \text{if } p \in \text{Out}(G) \text{ and } m(p) = s_M(e) \\ s_M(e) & \text{otherwise} \end{cases}$$
$$t_H(e) = \begin{cases} b^i(p) & \text{if } p \in \text{In}(G) \text{ and } m(p) = t_M(e) \\ t_M(e) & \text{otherwise} \end{cases}$$

where $B := P_{\text{Out}(G)} + P_{\text{In}(G)}$ is the boundary of G and $b : B \to G$ is the corresponding boundary map. Recall from Definition 5.13 that $b^o : \text{Out}(G) \to B$ is the coproduct

injection of the outputs of G into the boundary, and, likewise, $b^i : \text{In}(G) \to B$ is the coproduct injection of inputs.

This definition removes all of G from M and reconnects edges that entered G with the corresponding point from the boundary of G.

We call the induced embedding $c : B \hookrightarrow H$ the *coboundary* of b with respect to m. When $m : G \to M$ is implicitly defined by the context, we omit the m from our subtraction notation and simply write $M - G$.

For this definition to be valid, we need to show that H is an open-graph; specifically, that the maps s_H and t_H are total, well defined and have at most one in-edge and one out-edge.

Proof. The source map s_H is total because the source of an edge $e \in E_H$ is in the image of m if and only if it is an output of G. Because m is mono, s_H is well defined, and similarly for t_H. The result is an open-graph because it has at most one in-edge and one out-edge for every edge-point: all edges in the image of m are removed, and at most one edge (in the same direction as a removed edge) is introduced. $\square$

Theorem 6.11. Subtractions by open-graphs without isolated points are S-adhesive pushout complements: given $H := M -_m G$, the diagram

$$
\begin{array}{ccc}
B & \xrightarrow{\ b\ } & G \\
{\scriptstyle c}\downarrow & & \downarrow{\scriptstyle m} \\
H & \xrightarrow{\ f\ } & M
\end{array}
\tag{5}
$$

is an S-adhesive pushout, where b is the boundary map of G and c is the coboundary of m.

Proof. First we show that b, c is boundary coherent. Since G contains no isolated points, the maps b and c are mono. By the definition of subtraction, for $p \in B$, if $b(p)$ is an input, then $c(p)$ is an output. Similarly, if $b(p)$ is an output, $c(p)$ is an input. So b, c satisfies the boundary coherence condition.

The pushout of b and c is the result of identifying the boundary of G with its coboundary in H. By case analysis, for an arbitrary point or edge in the open-graph that results from the subtraction M', we get that M' is isomorphic to M, and for the induced embedding m' of G into M', the following diagram commutes:

$$
\begin{array}{ccc}
G & \xrightarrow{\ m\ } & M \\
{\scriptstyle m'}\downarrow & \nearrow{\scriptstyle \cong} & \\
M' & &
\end{array}
$$

So diagram (5) is also a pushout square for some (uniquely defined) f. $\square$
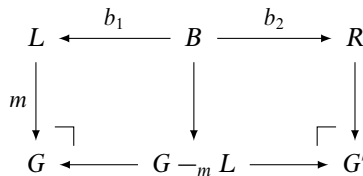
Note that because B in the above theorem is a point-graph, (b, c) defines a plugging, and $H := M -_m G$ is the uniquely determined open-graph such that $G +^*_{b,c} H \cong M$.

In terms of graph transformations (Ehrig *et al.* 2006), subtraction can be understood as a definition for constructing the context graph, while also ensuring that the result is a well-formed open-graph.

7. Rewriting with open-graphs

We will now introduce rewrite rules for open-graphs, and show how they can be applied and under what conditions they can commute with merging and plugging, and how rewrites can themselves be composed. Finally, we present a rewrite system called edge-homeomorphism that lets us ignore intermediate edge-points.

Definition 7.1 (rewrite rule and rewriting). A span $L \xleftarrow{b_1} B \xrightarrow{b_2} R$ in which L and R share the same boundary B by monos b_1 and b_2 is called a *rewrite rule* and is written $L \multimap_{b_1,b_2} R$. The rewrite rule is said to *rewrite* G to G' at a matching $m : L \to G$ when G' is defined according to the following double pushout:

$$
\begin{array}{ccccc}
L & \xleftarrow{\;b_1\;} & B & \xrightarrow{\;b_2\;} & R \\
{\scriptstyle m}\big\downarrow & & \big\downarrow & & \big\downarrow \\
G & \longleftarrow & G -_m L & \longrightarrow & G'
\end{array}
$$

The left pushout serves to compute the subtraction $G -_m L$, and the right pushout computes the rewritten open-graph G', which we shall also write as $G[L \multimap_{b_1,b_2} R]_m$.

Note that because we require a rule to be a span of monos, there can be no isolated points in L or R; the boundary map is 2–1 on isolated points.

Example 7.2 (circles). We now return to the challenging example introduced at the end of Section 2. We will rewrite the open-graph
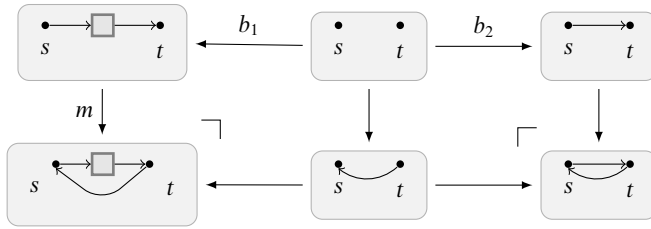

by
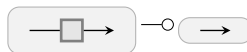

to get

The pushout construction for this rewrite is

Note that the above rewrite contains additional intermediate edge-points. Informally, these are intended to be treated as part of the edge. We will formalise this idea in Section 7.3 by introducing rewrite rules that can insert or remove these intermediate edge-points.
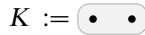
7.1. *Compatibility*

It may initially be surprising to realise that certain pushouts can prohibit certain rewrites. For example consider the following.

Example 7.3. Let

$$G := \boxed{\bullet}$$

$$H := \boxed{\bullet\!-\!\boxed{v}\!\rightarrow\!\bullet}.$$

For

$$K := \boxed{\bullet \quad \bullet}$$
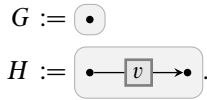
we can find maps

$$f : K \to G$$
$$g : K \to H$$

such that the S-adhesive pushout

$$G +_{f,g} H := \boxed{\bullet\!\mathrel{\reflectbox{$\rightarrow$}}\!\boxed{v}}$$
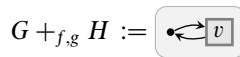
While the left-hand side of the rewrite

$$\boxed{\bullet\!-\!\boxed{v}\!\rightarrow\!\bullet} -\!\circ\!\boxed{\bullet\!\longrightarrow\!\bullet}$$

matches H, it does not match $G +_{f,g} H$.

Note that the pushout of the above example is not a plugging since f and g are not matchings (they are not mono). We will be primarily concerned with pluggings, which, as we shall prove, are better behaved. We will first provide a precise definition of what it means for a plugging and a rewrite to be compatible.

Definition 7.4 (compatibility). A plugging $G +_{p,q}^* H$ and a rewrite $G[L\!-\!\circ R]_m$ are said to be *compatible* when there exists a map $\widehat{p}$ and a matching $\widehat{m}$ such that $(\widehat{p}, q)$ is a plugging and
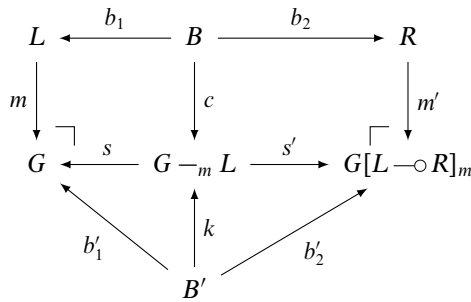
$$G[L\!-\!\circ R]_m +_{\widehat{p},q}^* H \cong (G +_{p,q}^* H)[L\!-\!\circ R]_{\widehat{m}}.$$

We can actually show that *all* pluggings and rewrites are compatible, but before we prove this important theorem, we will first show that the boundary of an open-graph is invariant under rewriting.

Theorem 7.5. Rewriting preserves the boundary of an open-graph. Specifically, let the top two squares of the following diagram define the rewrite $G[L\!-\!\circ R]_m$:

Then there exists a map k and a span of boundary maps b_1', b_2' making the bottom two triangles commute.

Proof. Let B' be the boundary of G and b_1' be its inclusion into G. We first show that the boundary of G is in the image of s by considering the definition of subtraction. If an arbitrary point x in B' is not in the image of m, then it is still in $G -_m L$. If it is in the image of m, then it must be in the boundary of L in G. Since a copy of this boundary is in $G -_m L$, we know x must be in the image of s. Thus, for all x in B', there exists x' in $G -_m L$ such that $s(x') = x$. Since s is mono (and hence injective), x' is unique, so let k be defined as s^{-1}: the map sending x to x'. Finally, let $b_2' = s'kb_1'$.

It now suffices to show that b_2' is a boundary map. If x is an input of G, then there are two cases for $k(x)$: it is either still an input or it is an isolated point. In the latter case, it must come from an input of L, and hence an input of R. Thus $s'(k(x))$ is an input in $G[L\!-\!\circ R]_m$. The same result follows similarly for outputs. By case analysis, $s'k$ covers the boundary of $G[L\!-\!\circ R]_m$, so b_2' is also a boundary map. $\square$

We can now use this theorem and Theorem 4.18 to show not only that pluggings and rewrites are compatible, but explicitly define the maps $\widehat{m}$ and $\widehat{p}$ used in Definition 7.4.

Theorem 7.6 (rewriting and plugging are compatible). Given a plugging

$$(p : K \to G, q : K \to H)$$

and a matching $m : L \to G$ of a rewrite rule $L \multimap R$, let i be the embedding of G into $G +^*_{p,q} H$. Then there exists $\widehat{p}$ such that $(\widehat{p}, q)$ is a plugging, $\widehat{m} := im$ is a matching and

$$G[L \multimap R]_m +^*_{\widehat{p},q} H \cong (G +^*_{p,q} H)[L \multimap R]_{im}.$$

Proof. Since i and m are matchings, so is im. Since p, q is a plugging, the map p factors through the boundary map $b'_1 : B' \to G$. Let r be a map such that $p = b'_1 r$. For k and b'_2 defined as in Theorem 7.5, let $p' = kr$ and $\widehat{p} = b'_2 r$. We know that the following diagram commutes:
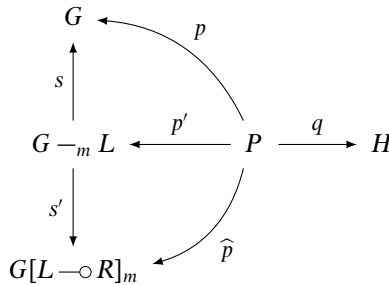

If $p(x)$ is an input, then $p'(x)$ and $\widehat{p}(x)$ are both inputs, and similarly for outputs. Therefore, (p, q), (p', q) and $(\widehat{p}, q)$ are all boundary-coherent spans, and thus S-adhesive spans. The result then follows from Theorem 4.18. $\qquad\square$

7.2. *Composition of rewrites*

Definition 7.7 (extension). Given an open-graph G and a rewrite rule $r := L \multimap R$, when r rewrites G to $G[L \multimap R]_m$, the rewrite rule $G \multimap_{b_1, b_2} G[L \multimap R]_m$ is said to be the *extension* of r by m, and written $r^{\uparrow m}$.

Note that this is a well-defined rewrite rule because the boundary span b_1, b_2 is uniquely defined by Theorem 7.5.

Example 7.8. Returning to Example 7.2, the extension of this rewrite is the span
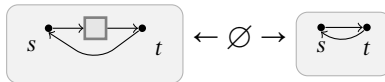
where the shared boundary is the empty open-graph, denoted by $\varnothing$.

Extension provides a construction of the rewrite relation $\longrightarrow_S$ for open-graphs. That is, $G \longrightarrow_S H$ precisely when there exists a rule in S that can be extended to $G \multimap H$.

We now show how rules can be directly combined using the underlying operations on open-graphs. We first note that any rewrite rule $L \multimap_{b_1, b_2} R$ has an opposite rewrite $R \multimap_{b_2, b_1} L$ given by flipping the span around. Also, for two rewrite rules $L \multimap_{b_1, b_2} R$ and $R \multimap_{b_2, b_3} R'$, we can, by abuse of notation, assume they are both spans over the same boundary graph, and write $L \multimap R \multimap R'$ for the rule $L \multimap_{b_1, b_3} R'$.

Definition 7.9 (sequential composition). Given rewrite rules

$$r_1 := L_1 \multimap R_1$$
$$r_2 := L_2 \multimap R_2$$

and a merged open-graph $M := R_1 +_{k_1,k_2} L_2$, the *sequential composition* of r_1 and r_2 at k_1, k_2 is the rewrite rule defined by

$$(r_1 \, ;_{k_1,k_2} r_2) := (M[R_1 \multimap L_1]_{m_1}) \multimap_{b_1,b_2} (M[L_2 \multimap R_2]_{m_2}),$$

where m_1 is the embedding of R_1 into M, m_2 is the embedding of L_2 into M and (b_1, b_2) is the following boundary span induced by two applications of Theorem 7.5:
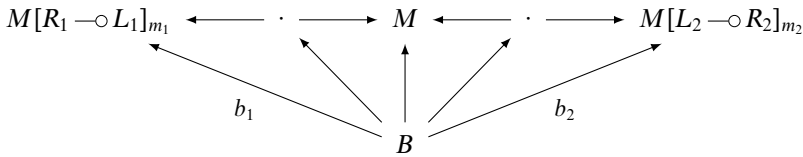
Sequential composition, unlike extension, is a direct operation on two rewrites to produce a new rewrite. This provides an algorithm for deriving new graphical equations, as we did in Section 2. Sequential composition is correct in the sense that it does nothing more than the reflexive, transitive closure of the rewrite relation $\overset{*}{\rhd}_{\mathbb{S}}$.

Theorem 7.10 (soundness). If $(r_1 \, ;_{k_1,k_2} r_2) := G \multimap G'$ is a rewrite, there exists an open-graph M and matchings m_1 and m_2 such that $G \xrightarrow{r_1 \uparrow m_1} M \xrightarrow{r_2 \uparrow m_2} G'$.

Proof. Let $r_1 := L_1 \multimap R_1$ and $r_2 := L_2 \multimap R_2$. Let M be exactly $R_1 +_{k_1,k_2} L_2$. The embedding of L_2 into M defines m_2. We now have to prove that there is an m_1 such that

$$M[R_1 \multimap L_1]_{m_1'}[L_1 \multimap R_1]_{m_1} \cong M,$$

where m_1' is the embedding of R_1 into M. This follows directly from expanding the equation into subtractions and mergings, and then recalling that subtractions are pushout complements. $\square$

Sequential composition of rewrites is also complete in the sense that a number of rewrites under different extensions, which are composed by their spans, can always be sequentially composed under a single extension.

Theorem 7.11 (completeness). If $M_1 \xrightarrow{r_1 \uparrow m_1} M_2 \xrightarrow{r_2 \uparrow m_2} M_3$, there exists an m' and k_1, k_2 such that

$$(r_1 \, ;_{k_1,k_2} r_2)^{\uparrow m'} : M_1 \multimap M_3.$$

Proof. Let $r_1 := L_1 \multimap R_1$ and $r_2 := L_2 \multimap R_2$. There is a matching of both R_1 and L_2 in M_2. The overlap of these matchings forms an open-graph K that defines the boundary coherent pair k_1, k_2 of K into R_1 and L_2, respectively. We then have

$$M_2 \cong (R_1 +_{p_1,p_2}^* L_2') +_{q_1,q_2}^* M_2',$$

where

$$L_2' := L_2 - K$$
$$M_2' := (M_2 - R_1) - L_2'.$$

Thus

$$M_1 \cong (L_1 +^*_{p_1,p_2} L_2') +^*_{q_1,q_2} M_2',$$

and m' is simply the embedding of $L_1 +^*_{p_1,p_2} L_2'$ into M_1. $\qquad\square$

7.3. Edge-homeomorphism

Although we have defined everything discretely so far, open-graphs admit a topological interpretation. Edges can be thought of as copies of the unit interval $[0, 1] \subset \mathbb{R}$ considered as an oriented manifold. Vertices are distinguished points, to which we ascribe semantic meaning, and edges represent 'gluing' intervals end-to-end, or gluing a vertex on to one edge of an interval. We now briefly elaborate on this idea before introducing a rewrite rule to act in a way analogously to homeomorphism.

Definition 7.12. For an open-graph G, a *wire* W in G is a set of connected edge-points, which contains at least one edge and may also include vertices at its start and end. If a vertex is connected to either end of W in G, it is called an *endpoint* of W.

As open-graphs, wires can be chains or circles. For any wire W, we can define an (oriented) manifold $M(W)$ as a quotient over the disjoint union of real unit intervals $\coprod [0, 1]_e$ indexed by the edges e in W. Whenever there are two edges e_1 and e_2 in W with $t(e_1) = s(e_2)$, we identify $1 \in [0, 1]_{e_1}$ with $0 \in [0, 1]_{e_2}$. The unit intervals $[0, 1]_e$ then form a collection of charts for $M(W)$ and give an orientation, so $M(W)$ forms an oriented manifold.

Definition 7.13. Two open-graphs G and G' are said to be *edge-homeomorphic* if G' can be obtained from G by replacing any wire W with a new wire W' where there exists a homeomorphism of oriented manifolds $M(W) \cong M(W')$.

This topological intuition is encoded discretely in open-graphs as a rewrite system called *edge-homeomorphism*.

Definition 7.14 (edge-homeomorphism). The following rewrite system $\mathbb{H}$ is called *edge-homeomorphism*:
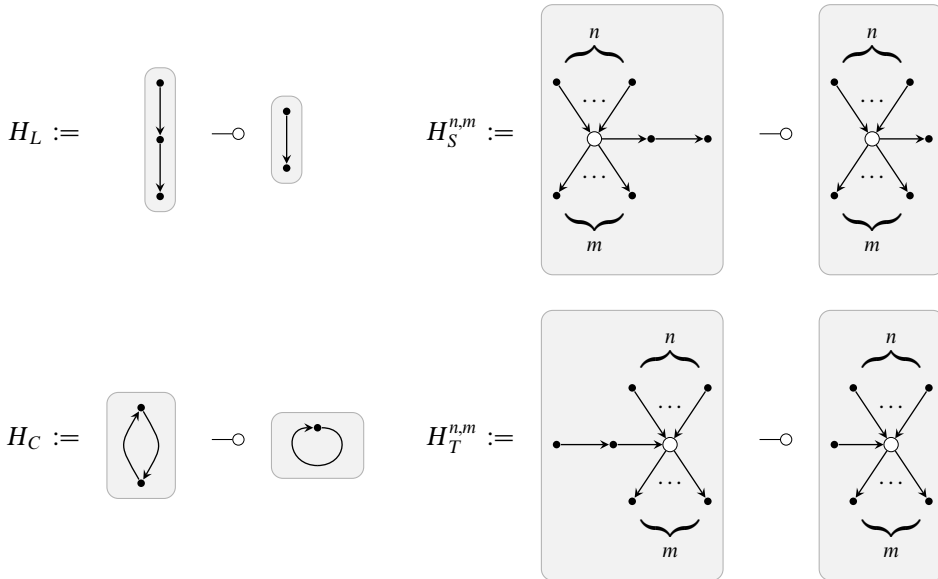
Applying edge-homeomorphism rewrites to an open-graph, from left to right, is called *contracting*. If G rewrites to H using zero or more edge-homeomorphism rewrites, we say H is an *edge contraction* of G. Applying them from right to left is called *expanding*. Edge-homeomorphism allows arbitrarily many edge-points to be inserted and removed from paths of connected edge-points.

Lemma 7.15. The rewrite system $\mathbb{H}$ is confluent (up to graph isomorphism) and terminating.

Proof. Termination comes from observing that each contraction of a morphism decreases the number of edge-points. To show confluence, we suppose we apply a rewrite that contracts away an edge e_1 in G to get a graph G'. Then, if we apply a different rewrite that contracts away a different edge e_2 in G to get a graph G'', we need to show there exist rewrites to make G' and G'' isomorphic. If e_1 and e_2 are both part of the same circle, consisting of k edges, then G' G'' will be the same open-graph as G, but with that circle now containing $k - 1$ edges. Thus $G' \cong G''$. Otherwise, there will always exist rewrites to contract away e_2 in G' and e_1 in G'', making the two open-graphs isomorphic. $\square$

Considering open-graphs modulo edge-homeomorphism corresponds to ignoring the intermediate edge-points. Returning to Example 7.2, the resulting circle with two edge-points can now be contracted to a circle with a single edge-point.

8. Typed open-graphs

We now generalise our definition of open-graphs by showing how they can be parametrised by a 'graphical signature' to form *typed open-graphs*. The graphical signature defines the types and arities of vertices, as well as the types of edges that can be used. This generalised construction makes use of more sophisticated type-graphs, which can themselves be embedded into the basic case of open-graphs. This lets us build a selective adhesive functor through which typed open-graphs inherit properties for rewriting.

Definition 8.1 (graphical signature). For a fixed set O, let O^* be the set of finite lists of O. For another set A, a function $T : A \to O^* \times O^*$ is called a *graphical signature*. T should be thought of as a function assigning input and output types to each element in A.

Example 8.2. For instance, a function T defined as

$$T :: \begin{cases} f & \mapsto ([\,A, B, C\,], [\,D, E\,]) \\ g & \mapsto ([\,E, E\,], [\,B\,]) \end{cases}$$

can be visualised as a set of 'boxes':

$$T := \left\{ \begin{array}{c} \text{A B C} \quad \text{E E} \\ \boxed{f} \, , \, \boxed{g} \\ \text{D E} \quad \text{B} \end{array} \right\} \tag{6}$$
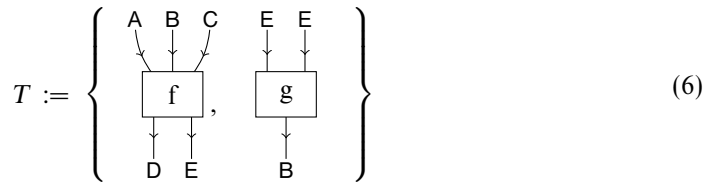
Remark 8.3. Graphical signatures are essentially what Selinger calls *monoidal signatures* (Selinger 2009) and Joyal and Street call *tensor schemes* (Joyal and Street 1991). We shall see the relationship between graphical signatures and the construction of free monoidal categories in Section 9.

Definition 8.4 (the typegraph of a graphical signature). For a graphical signature T, we form a typegraph $\mathscr{G}_T$, much like the typegraph $\mathscr{G}_2$ used to define open-graphs (Section 5). The typegraph has points $O + A$. Analagously to ϵ from $\mathscr{G}_2$, each point from O has a self-loop. These points identify the different types of edges and their corresponding edge-points. The points from A correspond to the different types of vertices. For each point a from A, $T(a)$ is a pair of words D, C defining the domain and codomain of a. These define the types of the inputs and outputs of a, respectively: for each input type d in D, $\mathscr{G}_T$ has an edge from d to a and for each output type c in C, $\mathscr{G}_T$ has an edge from a to c.

Example 8.5. The graphical signature T in Example 6 defines the typegraph $\mathscr{G}_T$:
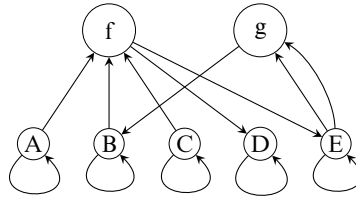
Definition 8.6 (typegraph notation). Let (G, τ) be a $\mathscr{G}_T$-typed graph in **Graph**$/\mathscr{G}_T$. Then points $p \in \tau^{-1}(O)$ in G are called *edge-points*; all other points, those from A, are called *vertices*.

The notion of local isomorphism lifts naturally to this set of typed graphs.

Definition 8.7. A $\mathscr{G}_T$-graph morphism $f : G \to H$ is called a *local isomorphism* when for every vertex $v \in G$, the edge function of f restricts to a bijection on the edge neighbourhood of v:

$$f^v : N(v) \overset{\sim}{\to} N(f(v)).$$

Example 8.8. Let τ be the (typed) graph on the left of the following diagram, and observe that its typing is identified by τ, depicted on vertices by the dotted arcs:
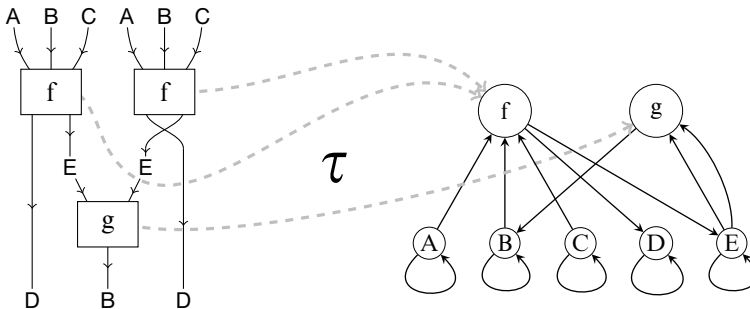
Note that, in addition to being an arrow in **Graph**, τ is a (terminal) arrow in **Graph**$/\mathscr{G}_T$, $\tau : (G, \tau) \to (\mathscr{G}_T, 1_{\mathscr{G}_T})$:

So we can ask that τ be a local isomorphism as in Definition 8.7.

We now consider a category $(\mathbf{Graph}/\mathscr{G}_T)_{\cong}$ that is the full subcategory of **Graph**$/\mathscr{G}_T$ whose objects are pairs $(G, \tau : G \to \mathscr{G}_T)$, where τ, considered as an arrow in **Graph**$/\mathscr{G}_T$, is a local isomorphism. We now prove that this restriction on objects implies that arrows in $(\mathbf{Graph}/\mathscr{G}_T)_{\cong}$ are themselves local isomorphisms.

Lemma 8.9. Every arrow in $(\mathbf{Graph}/\mathscr{G}_T)_{\cong}$ is a local isomorphism.

Proof. Let (G, τ_G), (H, τ_H) be $\mathscr{G}_T$-graphs. By definition, τ_G and τ_H are both local isomorphisms. For any $f : (G, \tau_G) \to (H, \tau_H)$ in $\mathbf{Graph}/\mathscr{G}_T$, the following diagram commutes:

$$
\begin{array}{ccc}
G & \xrightarrow{\tau_G} & \mathscr{G}_T \\
\downarrow{\scriptstyle f} & \nearrow{\scriptstyle \tau_H} & \\
H & &
\end{array}
$$

Thus, for any v in G, we get the following triangle in $\mathbf{Set}$:

$$
\begin{array}{ccc}
N(v) & \xrightarrow{\tau_G^v} & N(\tau_G(v)) \\
\downarrow{\scriptstyle f^v} & \nearrow{\scriptstyle \tau_H^{f(v)}} & \\
N(f(v)) & &
\end{array}
$$

Since τ_G^v and $\tau_H^{f(v)}$ are both bijections, f^v is a bijection, so f is a local isomorphism. $\qquad\square$

Note that for any $\mathscr{G}_T$, there is a graph homomorphism $\kappa : \mathscr{G}_T \to \mathscr{G}_2$ sending every edge-point (a point in O) to ϵ and every vertex (a point in A) to V. Post-composing each object in $\mathbf{Graph}/\mathscr{G}_T$ with κ yields the forgetful functor:

$$U_\kappa : \mathbf{Graph}/\mathscr{G}_T \to \mathbf{Graph}/\mathscr{G}_2.$$

In particular, this sends an object $\tau : G \to \mathscr{G}_T$ in $\mathbf{Graph}/\mathscr{G}_T$ to an object $\kappa \circ \tau$ in $\mathbf{Graph}/\mathscr{G}_2$.

We can now define typed open-graphs, which are essentially the typed graphs of $\mathbf{Graph}/\mathscr{G}_T$ with the restriction that the wires may not branch.

Definition 8.10 (T-open-graph). A $\mathscr{G}_T$-typed graph G is called a T-open-graph if $U_\kappa(G) \in \mathbf{Graph}/\mathscr{G}_2$ is an open-graph. The category $\mathbf{OGraph}_T$ is the full subcategory of $(\mathbf{Graph}/\mathscr{G}_T)_{\cong}$ whose objects are T-open-graphs.

Note that the forgetful functor U_κ restricts to another, and more useful, functor:

$$U : \mathbf{OGraph}_T \to \mathbf{OGraph}.$$

Lemma 8.11. Monos in $\mathbf{OGraph}_T$ are injective maps.

Proof. Suppose $m : G \to H$ in $\mathbf{OGraph}_T$ is not injective. If m takes two distinct edges e_1 and e_2 to a single edge, we suppose the source of e_1 (and hence of e_2) is an edge-point. Then, since G is a T-open-graph, it must take two edge-points to a single edge-point in H. Otherwise, if we suppose it is a vertex, then, by local isomorphism, m must take two distinct vertices on to a single vertex. Thus it suffices to only consider points.

If m takes two distinct vertices v_1, v_2 in G to a single vertex in H, then let K be the subgraph of G consisting of just v_1 and its neighbourhood. If m takes two distinct edge-points to a single edge-point in H, then let K be a T-open-graph consisting of a single edge-point. In either case, there are at least two distinct maps $f, g : K \to G$ such that $mf = mg$. $\qquad\square$

Theorem 8.12. The embedding functor $S' : \mathbf{OGraph}_T \to \mathbf{Graph}/\mathscr{G}_T$ is a selective adhesive functor.

Proof. From Lemma 8.11, S' preserves monos. Creation of isomorphisms follows from the fact that all isomorphisms are local isomorphisms and the property of being a T-open-graph is invariant under isomorphism. The faithfulness and reflection of pushouts follows from it being a full subcategory embedding. $\qquad\square$

Definition 8.13 (boundary-coherence in $\mathbf{OGraph}_T$). A span $A \xleftarrow{f} B \xrightarrow{g} C$ in $\mathbf{OGraph}_T$ is said to be *boundary-coherent* if its image under U is boundary-coherent in $\mathbf{OGraph}$.

Theorem 8.14. Boundary-coherent spans in $\mathbf{OGraph}_T$ are S' adhesive.

Proof. We prove this property by using the two embeddings and two forgetful functors:

$$
\begin{array}{ccc}
\mathbf{OGraph}_T & \xrightarrow{\;S'\;} & \mathbf{Graph}/\mathscr{G}_T \\
{\scriptstyle U}\Big\downarrow & & \Big\downarrow{\scriptstyle U_\kappa} \\
\mathbf{OGraph} & \xrightarrow{\;S\;} & \mathbf{Graph}/\mathscr{G}_2
\end{array}
$$

Let f, g be a boundary-coherent span in $\mathbf{OGraph}_T$, and let the following square be its pushout in $(\mathbf{Graph}/\mathscr{G}_T)_{\cong}$:

$$
\begin{array}{ccc}
S'(A) & \xrightarrow{\;S'(f)\;} & S'(B) \\
{\scriptstyle S'(g)}\Big\downarrow & & \Big\downarrow{\scriptstyle p_2} \\
S'(C) & \xrightarrow{\;p_1\;} & D
\end{array}
$$

Since $\mathbf{OGraph}_T$ is a full subcategory of $\mathbf{Graph}/\mathscr{G}_T$, it suffices to show that D is in $\mathbf{OGraph}_T$. By definition, $U(f), U(g)$ is boundary-coherent and hence S-adhesive in $\mathbf{OGraph}$, so its pushout D' exists and S preserves it. $S(D')$ is a pushout of

$$
(SU(f), SU(g)) = (U_\kappa S'(f), U_\kappa S'(g)).
$$

$U_\kappa(D)$ is the pushout of the right-hand side, so by the uniqueness of pushouts, $S(D') \cong U_\kappa(D)$. Finally, D' is in $\mathbf{OGraph}$, so D is a T-open-graph in $\mathbf{OGraph}_T$. $\qquad\square$

Boundary maps are defined as in **OGraph**. The construction of subtraction carries over verbatim, and is preserved by U. Also, analogously to **OGraph**, the uniqueness of pushout complements follows from the adhesiveness of **Graph**$/\mathscr{G}_T$.

9. Monoidal theories

Plugging gives us a tool for composing open-graphs. We can take this a step further and discuss composing open-graphs in a *categorical* sense, using cospan categories over **OGraph** or **OGraph**$_T$. For our purposes, we shall focus on the latter.

For a graphical signature

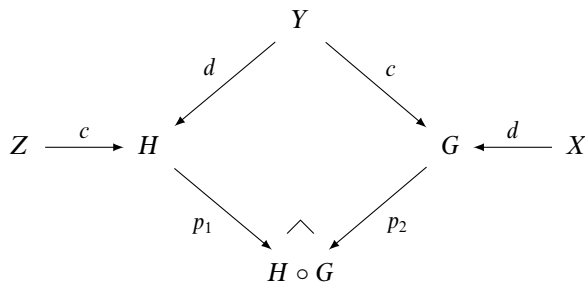$$T : A \to O^* \times O^*,$$

we define a precategory **DCsp(OGraph**$_T$**)** of *directed cospans* over **OGraph**$_T$ as a certain restriction of the usual cospan construction **Csp(**$\mathscr{C}$**)** on a category with pushouts. We then define a composition on **DCsp(OGraph**$_T$**)** that is associative and unital up to edge-homeomorphism. Thus, by forming a quotient of the hom-sets of **DCsp(OGraph**$_T$**)** over $\mathbb{H}$, we obtain a genuine category **DCsp(OGraph**$_T$**)**$/\!\!/ \mathbb{H}$.

Recall that a precategory $\mathscr{C}$ consists of a set (or proper class) of objects, and for each pair of objects, an associated hom-set. We make no assumptions about composition or identies. We define the precategory **DCsp(OGraph**$_T$**)** as follows. Its objects are words in O^*, considered as point-graphs in **OGraph**$_T$, equipped with a total ordering on points. An arrow $G : X \to Y$ in **OGraph**$_T$ is a cospan

$$Y \xrightarrow{\;c\;} G \xleftarrow{\;d\;} X$$

where G does not contain any isolated points[†], d is the inclusion of $\mathrm{In}(G) \cong X$, and c is the inclusion of $\mathrm{Out}(G) \cong Y$. In general, we shall write cospans from right-to-left to match '$\circ$'-style composition ordering. Also note that when there is no ambiguity, we shall use the letter G to refer both to the cospan and to the T-open-graph in the middle.

We can compose two cospans $G : X \to Y$ and $H : Y \to Z$ in **DCsp(OGraph**$_T$**)** by forming the pushout along Y.

Proposition 9.1. The cospan

$$Z \xrightarrow{\ p_1 \circ c\ } H \circ G \xleftarrow{\ p_2 \circ d\ } X$$

is an arrow in **DCsp(OGraph$_T$)**.

Proof. If G and H have no isolated points, then $H \circ G$ has no isolated points. Also, since the pushout over Y identifies all of the outputs of G with all of the inputs of H, the map $p_1 \circ c$ is an embedding of $Z \cong \mathrm{Out}(H) \cong \mathrm{Out}(H \circ G)$ and $p_2 \circ d$ is an embedding of $X \cong \mathrm{In}(G) \cong \mathrm{In}(H \circ G)$. □

It follows from the associativity of pushouts that this composition operation is associative up to boundary-preserving isomorphism. That is, for three cospans

$$Z \to K \leftarrow Y \to H \leftarrow X \to G \leftarrow W,$$

the following diagram commutes:
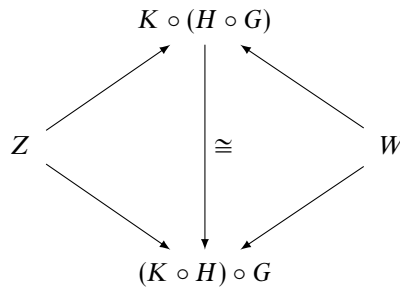
In the usual cospan construction, we would define 1_X as a cospan

$$X \xrightarrow{\ 1\ } X \xleftarrow{\ 1\ } X.$$

However, in the precategory of *directed* cospans, we have forbidden the middle T-open-graph from having any isolated points. Therefore, we consider cospans that behave as identities only *up to edge-homeomorphism*.

Definition 9.2 (identity T-open-graphs). For a point graph X, we construct Id_X as a T-open-graph with points $X + X$ and a single edge connecting $i_1(x)$ to $i_2(x)$ for each $x \in X$.

Example 9.3. Consider the following point graph:

$$X := \boxed{\quad \bullet \qquad \bullet \qquad \bullet \quad}$$

The T-open-graph Id_X is defined as

$$\mathrm{Id}_X := \boxed{\quad \updownarrow \qquad \updownarrow \qquad \updownarrow \quad}$$

Since $\text{In}(\text{Id}_X) \cong X$ and $\text{Out}(\text{Id}_X) \cong X$, the following cospan is an arrow $\text{Id}_X : X \to X$ in $\mathbf{DCsp}(\mathbf{OGraph}_T)$:

$$X \to \text{Id}_X \leftarrow X.$$

Note that $G \circ \text{Id}_X \not\cong G$ and $G \not\cong \text{Id}_Y \circ G$. However, G is equivalent to both of these T-open-graphs up to edge-homeomorphism. To see this, simply contract away the edges in $\text{Id}_X \subseteq G \circ \text{Id}_X$ or $\text{Id}_Y \subseteq \text{Id}_Y \circ G$ using one of the H-rules from Definition 7.14.

9.1. *Rewrite categories*

By passing from cospans of T-open-graphs to cospans of $\mathbb{H}$-equivalence classes of cospans, the composition $\circ$ becomes strictly associative and there exist identities for all objects. In particular, we can turn the precategory $\mathbf{DCsp}(\mathbf{OGraph}_T)$ into a geniune *category* $\mathbf{DCsp}(\mathbf{OGraph}_T)\!/\!\!/ \mathbb{H}$, called the *rewrite category* of $\mathbb{H}$. However, for $\circ$ to be well defined, we must first show that the composition operation does not depend on the choice of representative.

Lemma 9.4. Let the following cospan be an arrow in $\mathbf{DCsp}(\mathbf{OGraph}_T)$:

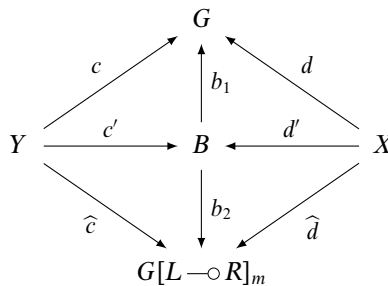$$Y \xrightarrow{c} G \xleftarrow{d} X.$$

Let m be a matching of a rewrite $L \multimap R$ on G, and let the induced rewrite be

$$G \xleftarrow{b_1} B \xrightarrow{b_2} G[L \multimap R]_m \tag{7}$$

where b_1 and b_2 are the boundary maps into G and $G[L \multimap R]_m$, respectively. Then there exists a unique $\widehat{c}$ and $\widehat{d}$ such that

$$Y \xrightarrow{\widehat{c}} G[L \multimap R]_m \xleftarrow{\widehat{d}} X$$

is an arrow in $\mathbf{DCsp}(\mathbf{OGraph}_T)$ and the following diagram commutes for some maps c' and d':

Proof. Since diagram (7) is a span of boundary maps, it restricts to a smaller span

$$G \xleftarrow{b_1'} \text{In}(G) \cong \text{In}(G[L \multimap R]_m) \xrightarrow{b_2'} G[L \multimap R]_m$$

where b_1' and b_2' are monos. Since the image of d is contained in the image of b_1', it factors uniquely through b_1' as $d = b_1' \circ d'$. Furthermore, $\hat{d} := b_2' \circ d'$ is the unique map making the above diagram commute. The construction follows similarly for c. $\square$

Definition 9.5 (rewriting on cospans). For a cospan

$$Y \xrightarrow{c} G \xleftarrow{d} X$$

in $\mathbf{DCsp}(\mathbf{OGraph}_T)$ and a matching m of a rewrite $L \multimap R$ on G, we write $G[L \multimap R]_m$ for the T-open-graph (and associated cospan) defined by Lemma 9.4.

Theorem 9.6. Let $G : A \to B$, $H : B \to C$ be cospans in $\mathbf{DCsp}(\mathbf{OGraph}_T)$, and m be a matching of a rule $L \multimap R$ on G. Then there exists a matching m' on $H \circ G$ such that

$$H \circ (G[L \multimap R]_m) \cong (H \circ G)[L \multimap R]_{m'}.$$

Similarly, for any cospan matching n on H, there exists n' such that

$$(H[L \multimap R]_n) \circ G \cong (H \circ G)[L \multimap R]_{n'}.$$

Proof. The result follows from Theorem 7.6 and Lemma 9.4. In both cases, m' and n' are formed by composing the original mapping with the inclusion of the matched T-open-graph into the (S'-adhesive) pushout. □

We will now define rewrite systems for cospans in the same way as we did for open-graphs.

Definition 9.7 (rewrite systems for cospans). Let $\mathbb{S}$ be a set of rewrite rules. We write $G \rightarrowtail_{\mathbb{S}} H$ if there exists a rule $L \multimap R$ in $\mathbb{S}$ and a cospan matching m such that $G[L \multimap R]_m \cong H$. Let $\overset{*}{\leftrightarrowtail}_{\mathbb{S}}$ be the closure of $\rightarrowtail_{\mathbb{S}}$ as an equivalence relation.

Let $\mathbb{H}$ be the typed version of the edge-homeomorphism rewrite system from Definition 7.14. This system consists of a line contraction rule $H_L(o)$ and a circle contraction rule $H_C(o)$ for each $o \in O$. It also has an input contraction rule $H_T^k(a)$ for each $a \in A$ and each input $k \in 1..N$ defined by $T(a)$, and similarly an output contraction rule $H_S^k(a)$. Note that when A and O are finite, this rewrite system is finite, unlike in the untyped case, where it is countably infinite.

Theorem 9.8. Up to edge-homeomorphism, the composition of cospans is associative and unital.

Proof. By the associativity of pushouts,

$$(I \circ H) \circ G \cong I \circ (H \circ G).$$

By definition, the relation $\overset{*}{\leftrightarrowtail}_{\mathbb{H}}$ subsumes cospan isomorphism. T-open-graphs of the form Id_X are identities with respect to composition. Since G contains no isolated points, every edge in $G \circ \mathrm{Id}_X$ that came from Id_X must be next to another edge. Therefore, one of the edge-homeomorphism rewrite rules can be used to contract it away. Thus, $G \circ \mathrm{Id}_X$ can be transformed back into G. Similarly, $\mathrm{Id}_Y \circ G \overset{*}{\leftrightarrowtail}_{\mathbb{H}} G$. □

We can now define rewrite categories.

Definition 9.9. Let $\mathbb{S}$ be a rewrite system containing $\mathbb{H}$. Then $\mathbf{DCsp}(\mathbf{OGraph}_T)/\!\!/\,\mathbb{S}$ is a category with:

— objects as (totally ordered) point-graphs;
— arrows are equivalence classes $\widetilde{G}$ of directed cospans, up to $\overset{*}{\lhd\!-\!\rhd}_\mathbb{S}$;
— composition defined by pushout of cospans; and
— identities as cospans $X \to \mathrm{Id}_X \leftarrow X$ for all X.

Theorem 9.10. All rewrite categories are symmetric monoidal categories.

Proof. The monoidal product is defined on objects and arrows by the disjoint union of T-open-graphs. Let

$$\widetilde{G} : X \to Y$$
$$\widetilde{H} : X' \to Y'$$

be arrows in $\mathbf{DCsp}(\mathbf{OGraph}_T)/\!\!/\,\mathbb{S}$ represented by cospans G and H, respectively. Then

$$\widetilde{G} \otimes \widetilde{H} : X \otimes X' \to Y \otimes Y'$$

is an arrow represented by the cospan

$$Y + Y' \to G + H \leftarrow X + X'.$$

Associativity follows from the associativity of coproducts in **OGraph**. Symmetry maps are defined as follows. Let $s : X + Y \to Y + X$ be the usual swap map for coproducts in **OGraph**. Then the swap map $\sigma : X \otimes Y \to Y \otimes X$ in $\mathbf{DCsp}(\mathbf{OGraph}_T)/\!\!/\,\mathbb{S}$ is represented by the cospan

$$Y + X \to \mathrm{Id}_{Y+X} \leftarrow Y + X \overset{s}{\leftarrow} X + Y.$$

The equations $\sigma \circ \sigma = \mathrm{Id}$ and $(\widetilde{G} \otimes \widetilde{H}) \circ \sigma = \sigma \circ (\widetilde{H} \otimes \widetilde{G})$ follow from T-open-graph isomorphism and rewriting with the edge-homeomorphism rules. $\square$

We shall now focus on the special case of $\mathbf{DCsp}(\mathbf{OGraph}_T)/\!\!/\,\mathbb{H}$ and show that it defines the *free* (symmetric or traced) modoidal category over a graphical signature T.

9.2. *Free monoidal categories*

For a class O, let O^* be the class of finite words formed from elements of O.

Definition 9.11 (monoidal precategory). A *monoidal precategory* consists of a class of *primitive objects* O and for every pair $v, w \in \mathrm{ob}\mathscr{M} := O^*$ a set $\mathrm{hom}(v, w)$ of arrows. A monoidal prefunctor $F : \mathscr{M} \to \mathscr{N}$ consists of a monoid homomorphism $\mathrm{ob}\mathscr{M} \to \mathrm{ob}\mathscr{N}$ and for every hom-set a function $\mathrm{hom}(v, w) \to \mathrm{hom}(Fv, Fw)$. The category of monoidal precategories and monoidal prefunctors is called **MonPreCat**.

Note that monoidal precategories do not necessarily have composition or identities, and the 'monoidal product' is only defined for objects. Monoidal categories and graphical signatures are both cases of monoidal precategories. In the case of a graphical signature

$T : A \to O^* \times O^*$, the class of primitive objects is O, and for any pair of words $v, w \in O^*$, the hom-set is formed from the inverse image of T:

$$\text{hom}(v, w) := T^{-1}(v, w) \subseteq A.$$

Definition 9.12. Let $\mathbf{TSMC}(T) := \mathbf{DCsp}(\mathbf{OGraph}_T) /\!\!/ \,\mathbb{H}$. Let $\mathbf{SMC}(T)$ be the subcategory of $\mathbf{TSMC}(T)$ where every T-open-graph in the middle of a cospan is directed acyclic.

$\mathbf{SMC}(T)$ has the property that no T-open-graphs contain 'feedback loops'. Note that T, as a monoidal precategory, embeds canonically into $\mathbf{SMC}(T)$, and hence into $\mathbf{TSMC}(T)$.

Theorem 9.13. $\mathbf{SMC}(T)$ is the free symmetric monoidal category of T. That is, for any symmetric monoidal category $\mathscr{V}$, any monoidal prefunctor $F : T \to \mathscr{V}$ extends uniquely to a symmetric monoidal functor from $\mathbf{SMC}(T)$. For the embedding of $T \hookrightarrow \mathbf{SMC}(T)$, there exists a unique monoidal functor $\hat{F}$ making the following diagram commute:
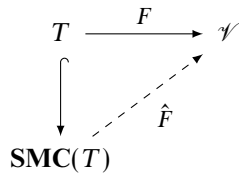
We can prove the above theorem using the geometric characterisation of symmetric monoidal categories given in Joyal and Street (1991). The details of this proof are given in Appendix A.

Definition 9.14 (trace operator). For objects A, B and C of $\mathbf{TSMC}(T)$, a *trace operator* is defined to be a function

$$tr_{A,C}^B(-) : \text{hom}_{\mathbf{TSMC}(T)}(A \otimes B, C \otimes B) \to \text{hom}_{\mathbf{TSMC}(T)}(A, C).$$

Intuitively, this introduces edges that connect from B in the codomain to B in the domain.
We first construct the identity graph as in Definition 9.2:

$$B \xrightarrow{o} \text{Id}_B \xleftarrow{i} B$$

Let $\widetilde{G} : A \otimes B \to C \otimes B$ be an arrow in $\mathbf{TSMC}(T)$, which is represented by a cospan G. The arrows of the cospan G factor over the coproduct $A + B$ and $C + B$, so we can write them as

$$C + B \xrightarrow{[c_1, c_2]} G \xleftarrow{[d_1, d_2]} A + B$$

for arrows $c_1 : C \to G$, $c_2 : B \to G$, and so on.

We now perform the following (boundary-coherent) pushout of G and Id_B, for $[d_2, c_2]$ the induced map from the coproduct $B + B$:
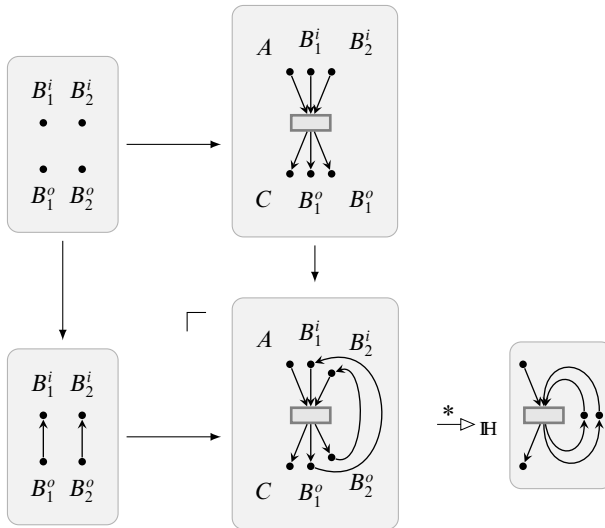
$$
\begin{array}{ccc}
B + B & \xrightarrow{[d_2, c_2]} & G \\
{\scriptstyle [o, i]}\downarrow & & \downarrow{\scriptstyle p_1} \\
\mathrm{Id}_B & \xrightarrow[p_2]{} & G'
\end{array}
$$

p_1 is mono because $[o, i]$ is, so let the following be a new cospan G':

$$
C \xrightarrow{p_1 c_1} G' \xleftarrow{p_1 d_1} A.
$$

Define $tr^B_{A,C}(\widetilde{G}) := \widetilde{G'}$. Since plugging and rewriting are compatible, this does not depend on the choice of representative G.

Example 9.15. The following diagram is an illustration of applying a trace operator:

This provides a natural way of working with traced symmetric monoidal categories, and subsequently compact closed categories. We conjecture that this is, in fact, a free construction of traced symmetric monoidal categories.

Conjecture 9.16. The trace operator of Definition 9.14 gives $\mathbf{TSMC}(T)$ the structure of the free traced symmetric monoidal category over T.

Remark 9.17 (free categories and isolated points). It is now possible to see why we disallow isolated points in the category $\mathbf{DCsp}(\mathbf{OGraph}_T)$. Consider the simplest case of a cospan involving an isolated point in the middle. Let P be the graph consisting of just one isolated point. Now consider composing the cospan

$$
P \xrightarrow{1} P \xleftarrow{1} P
$$

with the cospan Id_P. Since rewrite rules must be spans of monomorphisms to ensure that the rewrite process is well defined, they cannot contain isolated points in either the left- or right-hand side. Thus there cannot be a rule that relates P and $P \circ \mathrm{Id}_P \cong \mathrm{Id}_P$, so Id_P would no longer be an identity if isolated points were allowed. Worse still, we would have two cospans, P and Id_P, that should be interpreted as identity maps in the free category, and which are incomparable.

9.3. *PROPs*

PROPs, or PROduct categories with Permutations, are a convenient way of describing symmetric monoidal algebraic structures *internal* to some monoidal category $\mathscr{V}$.

Definition 9.18. A PROP is a symmetric monoidal category whose objects are the natural numbers and where the tensor product is given by addition.

Examples of PROPs are the (skeletal) category $\mathbb{F}$ of finite sets and functions, $\mathbf{Csp}(\mathbb{F})$ of cospans of finite sets, with composition as pushout, $\mathbf{Mat}(\mathbb{N})$ whose objects are natural numbers m, n and whose arrows are $m \times n$ matrices of natural numbers, and $\mathbf{Mat}(\mathbb{Z})$ the same for integers.

PROPs are interesting because they define categories of algebras.

Definition 9.19. For a PROP $\mathbb{P}$ and some fixed symmetric monoidal category $\mathscr{V}$, the category $\mathbb{P}$-Alg of $\mathbb{P}$-algebras has as objects strict symmetric monoidal functors $\mathbb{P} \to \mathscr{V}$ and has as arrows monoidal natural transformations.

As their name suggests, algebras of PROPs represent internal algebraic structures. For instance, the algebras of $\mathbb{F}$, $\mathbf{Csp}(\mathbb{F})$, $\mathbf{Mat}(\mathbb{N})$ and $\mathbf{Mat}(\mathbb{Z})$ in $\mathscr{V}$ are internal monoids, special Frobenius algebras, bialgebras and Hopf algebras, respectively.

PROPs can be combined with each other in much the same way as monads using *distributive laws* (Lack 2004), and even more flexible *interaction theories*, like the one used in Coecke and Duncan (2009).

A rich class of PROPs can be obtained from the rewrite categories defined in Section 9.1. Consider a typed graph category made from a 'single-sorted' graphical signature

$$T : A \to \{\bullet\}^* \times \{\bullet\}^*.$$

Hence, the objects of $\mathbf{DCsp}(\mathbf{OGraph}_T)$ are point graphs containing n isolated points of type '$\bullet$', which we can represent by the natural numbers. Since the monoidal product on objects is the disjoint union, $m \otimes n = m + n$.

Let $\mathbb{E}$ be some graphical theory, expressed as a rewrite system. Then, for $\mathbb{H}$ the edge-homeomorphism rewrite system, we can form the combined system $\mathbb{E} + \mathbb{H}$, and the rewrite category

$$\mathscr{E} := \mathbf{DCsp}(\mathbf{OGraph}_T) /\!/ (\mathbb{E} + \mathbb{H}).$$

The algebras of $\mathscr{E}$ will be structures in $\mathscr{V}$ that satisfy precisely the identities given graphically by E. By expanding the graphical signature T, this procedure generalises naturally from PROPs to multi-sorted monoidal theories. Taking $\mathscr{V}$ to be some concrete

category like **Vect$_\mathbb{C}$**, this formalises the notion of *concrete models* for some graphical theory.

10. Conclusions and further work

We have presented a theory of open-graphs to support graphical reasoning about computational processes. These graphs are visualised with an interface made of half-edges that enter or leave the graph. We formalised this by introducing a notion of intermediate points that occur along an edge or 'wire'. This allows a single wire to be cut into arbitrarily many smaller wires, and, conversely, supports composition by plugging wires together. We also described methods supporting graphical rewriting using the so-called double pushout approach, and showed how graphical rewriting rules can themselves be composed. We then formalised the relationship between graphs that are 'semantically' the same by defining a graph rewrite system called *edge-homeomorphism*, by analogy with homeomorphism in topological spaces.

We then generalised our construction of open-graphs to work with many *types* of vertices and wires. In particular, we parameterised open-graphs using a graphical signature that provides the typing constraints for composing graphs. The typing of open-graphs lets us express many kinds of processes, notably those with distinguished inputs and outputs. Building on graphical signatures, we then showed that cospans over typed open-graphs, modulo edge-homeomorphism, form free symmetric monoidal categories over a set of generators. By taking richer rewrite systems, we can obtain a large and interesting class of monoidal theory categories, including PROPs. In this way, we have provided a general method of reasoning about a wide variety of graphical theories.

The constructions presented here have deliberately been kept finitary and decidable for the case of finite open-graphs. This is with an eye to implementation of graphical reasoning software, which would form a conceptual bridge to let us enjoy the intuitive power of graphical languages, while benefiting from rigorous, computer-assisted manipulation. In particular, our theory provides a platform for bringing techniques from rewriting, such as critical pair analysis and Knuth–Bendix completion (Knuth and Bendix 1970), to process-centric graphical languages and monoidal categories. An implementation of this work is already largely completed[†], although a proof that this does indeed implement the theory presented here remains for future work. Another area of further work is to extend this formalism to support pattern-graphs, as introduced in Dixon and Duncan (2009). More generally, we would like to be able to reason with graphs and rules that contain repeated or recursive structures.

Our construction of PROPs and free symmetric monoidal categories is also only a beginning. The graphical notation for traced symmetric monoidal categories introduced in, for example, Selinger (2009) gives us strong reason to believe that Conjecture 9.16 is correct. For a suitable definition of traced monoidal theories, generalising the definition of PROPs to the traced setting, we believe that the construction in Section 9.3 actually

[†] see http://dream.inf.ed.ac.uk/projects/quantomatic for details.

forms the *free* traced monoidal theory satisfying the equations reflected by a rewrite system.

On a more fundamental level, the notion of edge-homeomorphism suggests a deep and telling connection not only with topological graphs, but also with their more exotic cousins, topological *directed graphs*. This has previously only been explored in an *ad hoc* manner, but we believe it can be made fully formal using the notions of directed topological spaces, as presented in Grandis (2009) or Krishnan (2009). We feel that, in the context of such a presentation, the technical content of this paper will arise naturally as a discrete reflection of the deeper, topological theory.

Appendix A. Proof of freeness for SMC(T)

We shall prove Theorem 9.13 using the geometric characterisation of symmetric monoidal categories given in Joyal and Street (1991).

Theorem 9.13. SMC(T) is the free symmetric monoidal category of T. That is, for any symmetric monoidal category $\mathscr{V}$, any monoidal prefunctor $F : T \to \mathscr{V}$ extends uniquely to a symmetric monoidal functor from **SMC(T)**. For the embedding of $T \hookrightarrow$ **SMC(T)**, there exists a unique monoidal functor $\hat{F}$ making the following diagram commute:

$$
\begin{array}{ccc}
T & \xrightarrow{\ F\ } & \mathscr{V} \\
\downarrow & \nearrow & \\
\mathbf{SMC}(T) & \hat{F} &
\end{array}
$$
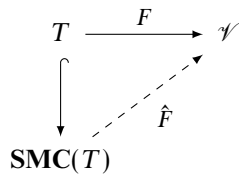
First, we recall several definitions from Joyal and Street (1991).

Definition A.1 (generalised topological graph). A *generalised topological graph* is a pair (G, G_0), where G is a Hausdorff space and G_0 is a discrete, closed subset where $G - G_0$ is isomorphic to a sum of open intervals $I_o := (0, 1) \subseteq \mathbb{R}$ and copies of S_1. The compactification of an open interval $I_o \subseteq G - G_0$ is called an *edge* $\hat{e}$. A copy of $S_1 \subseteq G - G_0$ is called a *circle* $\hat{c}$.

Note that all edges naturally embed in the compactification $\hat{G} \supseteq G$ obtained by adding endpoints to open edges.

Definition A.2 (polarised graph). A *polarised graph* is a tuple $\Gamma := (G, G_0, \omega, \pi)$, where ω assigns an orientation to each $\hat{e}$ and each circle $\hat{c}$ in (G, G_0). We can therefore define an input $\hat{e}(0)$ and an output $\hat{e}(1)$ for each edge. For each vertex $v \in G_0$, in(v) is the set of edges such that $\hat{e}(1) = v$ and out(v) is the set of edges such that $\hat{e}(0) = v$. Finally, π assigns to each v a total order on in(v) and out(v), called a *polarisation*. A polarised graph that contains no directed cycles is said to be *progressive*.

Polarised graphs come with a notion of boundary. Furthermore, we can put an ordering on this boundary.

Definition A.3 (boundary of a polarised graph). For a polarised graph $\Gamma := (G, G_0, \omega, \pi)$, $\partial\Gamma := \hat{G} - G$ is a discrete space called the *boundary* of Γ. Points in $\partial\Gamma$ that are the input of some edge are called *inputs* of Γ, and outputs of edges in $\partial\Gamma$ are called *outputs* of Γ. A polarised graph with a pair of total orders β_0 on its inputs and β_1 on its outputs is called an *anchored graph*.

Definition A.4 (valuation). For an anchored graph Γ and a monoidal precategory $\mathcal{M}$, a *valuation* v of Γ is a function v_0 that assigns an object of $\mathcal{M}$ to every edge in Γ and a function v_1 that assigns an arrow to every vertex in such a way that respects the domain and codomain of the arrows in $\mathcal{M}$. A map of anchored graphs with valuations $(\Gamma, v) \rightarrow (\Gamma', v')$ is a collection of maps that respect all of the structure of Γ and the valuations.

Since an anchored graph gives a total order to inputs and outputs, we can associate input and output words to a pair (Γ, v). Let $T : A \rightarrow O^* \times O^*$ be a graphical signature. $\mathbb{F}_S(T)$ is the category whose objects are words in O^*. For words v and w, arrows are isomorphism classes of progressive anchored graphs with valuations into T that have input word v and output word w.

Joyal and Street (1991) showed that $\mathbb{F}_S(T)$ is the free symmetric monoidal category over T. For the proof of Theorem 9.13 it suffices to show that a symmetric monoidal equivalence exists from $\mathbf{SMC}(T)$ to $\mathbb{F}_S(T)$.

We can now prove Theorem 9.13 by defining a geometric realisation functor $[\![-]\!]_T : \mathbf{SMC}(T) \rightarrow \mathbb{F}_S(T)$ that is identity-on-objects and showing it admits a (weak) inverse.

Proof. We let $\widetilde{G} : X \rightarrow Y$ be an arrow in $\mathbf{SMC}(T)$ and choose a directed cospan $Y \xrightarrow{c} G \xleftarrow{d} X$ of T-open-graphs to represent the equivalence class $\widetilde{G}$.

The category **Graph** sits inside the category of simplicial complexes, so there is a geometric realisation functor $[\![-]\!] : \mathbf{Graph} \rightarrow \mathbf{Top}$.

G is an element of the slice category over $\mathscr{G}_T$, so it comes with a map $\tau_G : G \rightarrow \mathscr{G}_T$. The underlying graph of G has an embedding of its boundary and its set of vertices. That is, there exist maps $b : X + Y \rightarrow G$ and $v : V \rightarrow G$ in **Graph**, where $X + Y$ and V are discrete graphs.

For $H := [\![G]\!] - [\![X + Y]\!]$ and $H_0 := [\![V]\!]$, (H, H_0) defines a generalised topological graph. Note that the compactification $\hat{H} = [\![G]\!]$. Since each edge (or circle) in $\hat{H}$ has an underlying directed chain (or cycle) of edge points, we can equip it with an orientation ω. Recall that edges adjacent to a vertex in $\mathscr{G}_T$ have a natural total order given by their word order in T. We can use this order to assign a polarisation π to the vertices in H_0. Thus (H, H_0, ω, π) defines a polarised graph. It is progressive precisely because G is directed-acyclic. The total order on X and Y induces a total order on the inputs and outputs of G, and hence total orders β_i, β_o on the inputs and outputs of the polarised graph. Thus $\Gamma = (H, H_0, \omega, \pi, \beta)$ is a progressive anchored graph. For Γ, a valuation v into T can clearly be deduced by the typing map $\tau_G : G \rightarrow \mathscr{G}_T$, so (Γ, v) is an arrow in $\mathbb{F}_S(T)$.

Let Γ' be the result of performing this construction on some other G' representing $\widetilde{G}$. Then G could be rewritten to G' by only merging or subdividing edges. The only step of

the construction that makes explicit reference to (internal) edge-points is the application of $[\![-]\!]$: **Graph** $\rightarrow$ **Top** to the underlying graphs of G and G'. This process forgets edge points, so $\Gamma' \cong \Gamma$. Also, for any G' that yields a progressive anchored graph $\Gamma' \cong \Gamma$, G' is simply another triangularisation of Γ, so G' rewrites to G using edge-homeomorphism.

This construction respects composition and the symmetric monoidal structure, so $[\![\widetilde{G}]\!]_T = \Gamma$ defines a symmetric monoidal functor into $\mathbb{F}_S(T)$. Furthermore, $[\![-]\!]_T$ admits a weak inverse by sending a progressive anchored graph Γ to the equivalence class $\widetilde{G}$ represented by *any* G such that the above construction performed on G yields a progressive anchored graph $\Gamma' \cong \Gamma$. $\qquad\Box$

Acknowledgements

References

Adamek, J., Herrlich, H. and Strecker, G. E. (2009) *Abstract and Concrete Categories: The Joy of Cats*, Dover.

Appelgate, H., Barr, M., Beck, J., Lawvere, F., Linton, F., Manes, E., Tierney, M. and Ulmer, F. (1969) Distributive laws. In: Seminar on Triples and Categorical Homology Theory. *Springer-Verlag Lecture Notes in Mathematics* **80** 119–140.

Baader, F. and Nipkow, T. (1998) *Term rewriting and all that*, Cambridge University Press.

Baldan, P., Corradini, A. and König, B. (2008) Unfolding graph transformation systems: Theory and applications to verification. In: Degano, P., de Nicola, R. and Meseguer, J. (eds.) Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday. *Springer-Verlag Lecture Notes in Computer Science* **5065** 16–36.

Coecke, B. and Duncan, R. (2008) Interacting quantum observables. In: Aceto, L. *et al.* (eds.) Automata, Languages and Programming: Proceedings ICALP 2008. *Springer-Verlag Lecture Notes in Computer Science* **5126** 298–310.

Coecke, B. and Duncan, R. (2009) Interacting quantum observables: Categorical algebra and diagrammatics. arXiv:0906.4725v1 [quant-ph].

Coecke, B. and Kissinger, A. (2010) The compositional structure of multipartite quantum entanglement. arXiv:1002.2540v2 [quant-ph].

Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theoretical Computer Science* **325** (1) 69–110.

Dixon, L. and Duncan, R. (2009) Graphical reasoning in compact closed categories for quantum computation. *AMAI* **56** (1) 20.

Dixon, L., Duncan, R. and Kissinger, A. (2010) Open graphs and computational reasoning. In: Proceedings Sixth Workshop on Developments in Computational Models: Causality, Computation, and Physics. *Electronic Proceedings in Theoretical Computer Science* **26** 169–180.

Ehrig, H., Ehrig, K., Prange, U. and Taentzer, G. (2006) *Fundamentals of Algebraic Graph Transformation*, Monographs in Theoretical Computer Science, EATCS Series, Springer-Verlag.

Ehrig, H., Pfender, M. and Schneider, H. J. (1973) Graph-grammars: An algebraic approach. In: *14th Annual Symposium on Switching and Automata Theory*, IEEE 167–180.

Girard, J.-Y. (1996) Proof-nets: The parallel syntax for proof-theory. In: *Logic and Algebra*, Marcel Dekker 97–124.

Grandis, M. (2009) *Directed Algebraic Topology: Models of Non-Reversible Worlds*, New Mathematical Monographs, Cambridge University Press.

Joyal, A. and Street, R. (1991) The geometry of tensor calculus I. *Advances in Mathematics* **88** 55–113.

Knuth, D. E. and Bendix, P. B. (1970) Simple word problems in universal algebras. In: *Computational Problems in Abstract Algebra*, Pergamon Press 263–297.

Krishnan, S. (2009) A convenient category of locally preordered spaces. *Applied Categorical Structures* **17** 445–466.

Lack, S. (2004) Composing props. *Theory and Applications of Categories* **13** (9) 147–163.

Lack, S. and Sobocinski, P. (2005) Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications* **39** (2) 522–546.

Lafont, Y. (1990) Interaction nets. In *POPL '90: Proceedings of the 17th ACM SIGPLAN–SIGACT symposium on Principles of programming languages* 95–108.

Lafont, Y. (2003) Towards an algebraic theory of boolean circuits. *Journal of Pure and Applied Algebra* **184** (2-3) 257–310.

Lafont, Y. (2010) Diagram rewriting and operads. In: Lecture Notes from the Thematic school: Operads CIRM, Luminy (Marseille) 20–25 April 2009.

Lafont, Y. and Rannou, P. (2008) Diagram rewriting for orthogonal matrices: A study of critical peaks. In: Voronkov, A. (ed.) Rewriting Techniques and Applications: Proceedings RTA'08. *Springer-Verlag Lecture Notes in Computer Science* **5117** 232–245.

Milner, R. (2006) Pure bigraphs: Structure and dynamics. *Information and computation* **204** (1) 60–122.

Penrose, R. (1971) Applications of negative dimensional tensors. In: Welsh, D. (ed.) *Combinatorial Mathematics and its Applications*, Academic Press 221–244.

Prange, U., Ehrig, H. and Lambers, L. (2008) Construction and properties of adhesive and weak adhesive high-level replacement categories. *Applications of Categorical Structures* **16** 365–388.

Selinger, P. (2009) A survey of graphical languages for monoidal categories. In: Coecke, B. (ed.) New Structures for Physics. *Springer-Verlag Lecture Notes in Physics* **813** 275–337.