Open Graphs and Monoidal Theories

Lucas Dixon
University of Edinburgh
ldixon@inf.ed.ac.uk

Aleks Kissinger
University of Oxford
alexander.kissinger@comlab.ox.ac.uk

Draft: May 22, 2018*

Abstract

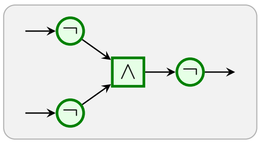
String diagrams are a powerful tool for reasoning about physical processes, logic circuits, tensor networks, and many other compositional structures. The distinguishing feature of these diagrams is that edges need not be connected to vertices at both ends, and these unconnected ends can be interpreted 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 discreet, finitary, and enjoys decidable algorithms for composition and rewriting. We also show how open-graphs can be parametrised by graphical signatures, similar to the monoidal signatures of Joyal and Street, which 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 handle 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 naturally abstract structure. In this paper, we will focus on "string diagrams", 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 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:

[*] This research was funded by EPSRC grant EPE/005713/1 and by a Clarendon Studentship. We would also like to thank Ross Duncan and Matvey Soloviev for their their many helpful discussions and remarks on this work.



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.

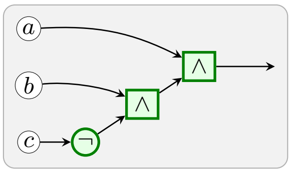
To formalise the process of rewriting, we use a well-behaved embedding of the category of open-graphs into its ambient category of typed graphs. This embedding is an instance of a more general notion which we introduce as *selective adhesive functors*. In particular, these functors reflect pushouts, so many results about pushouts in an adhesive category are true of so-called *adhesive pushouts*, i.e. 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 type graph from such a signature and form the category of *typed open-graphs* by slicing over this type graph. 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], to mechanise working with string diagrams.

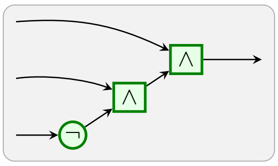
The rest of the paper is structured as follows. In section 2, we introduce and motivate graphical theories with boolean circuits and tensor networks. We also note key challenges in working with these systems using traditional graph-based methods. After reviewing some of these methods in section 3, we 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 conclude and discuss future work in section 10.

2 Motivating Examples

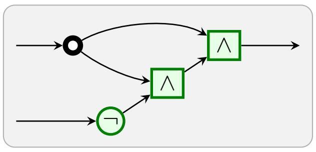
We introduce two examples here to motivate the use of open-graphs for computation. The first is the familiar language of boolean circuits. Boolean circuits are formed by taking basic logic gates and plugging them together. For instance, we can represent the logical expression “$a \wedge (b \wedge \neg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



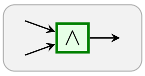
Notice that the output wire of this graph does not end at a vertex. We call this a *half-edge*. We can also represent inputs to a circuit as half-edges. In the above example, this removes the need to introduce the variables a, b, and c as inputs to the circuit. Instead, we represent the inputs as half-edges:



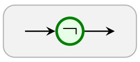
Now, suppose we wanted to introduce an expression like “$a \wedge (\neg a \wedge b)$”. We can do this without introducing explicitly named variables by introducing a “copy” operation.



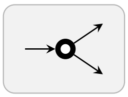
We can also introduce an explicit “ignore” operation that takes on input and produces no output. To sum up, our language has the following generators, where b is a boolean value.

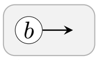


And

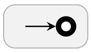


Not

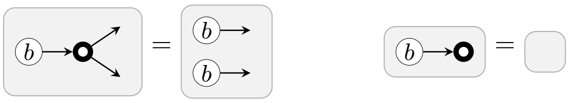

Copy



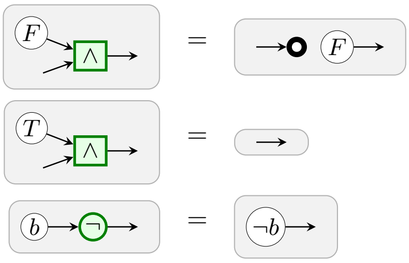
Boolean value


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 T. 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:

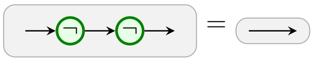


The axioms for conjunction (and-gates: ∧) and negation (not-gates: ¬) are:



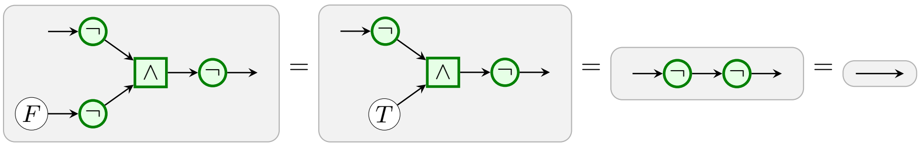
These rules characterise the computational aspects of boolean circuits. Applying the axioms from left to right can be used to evaluate the output of a circuit. The equations can also be used to simplify circuits.

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:



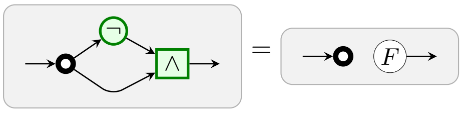
However, verification of such circuits can be done by exhaustive analysis directly in the graphical language: we can evaluate every combination of inputs to a graphical equation to see if the left- and right-hand sides always evaluate to the same result. This corresponds to a proof by exhaustive case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

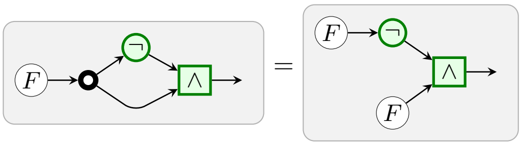


This proves that giving F to the compound or-gate is the same as the identity on the other input. Such derivations can be exponentially shorter than case-analysis. Moreover, rules in a derivation can simultaneously be applied to separate parts of a graph to parallelise a computation or derivation.

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 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 in graphical representations of quantum information, where copying can only happen in restricted situations.

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.

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

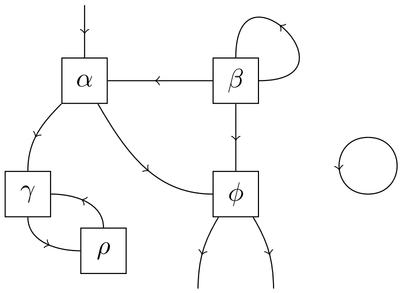
Tensors are written with subscript indices, which serve the purpose of inputs, and superscript indices which are outputs. Familiar examples of tensors are vectors, v^i and matrices, M_j^i. We can compose tensors by *contraction*, i.e. "summing together" a lower index and an upper index of the same dimension:

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

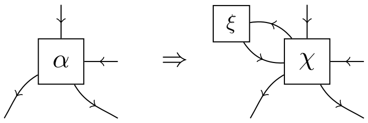
In order to simplify such expressions, we can use the Einstein summation convention, where any repeated indexes are assumed to be summed over. However, even with this convention, contraction expressions can get quite complex. Consider this expression, involving six tensors:

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

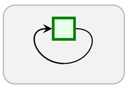
In order to understand this expression, one has to keep track of 11 indices, which makes computations time-consuming and error-prone. We can instead represent this expression using a graphical language introduced by Penrose [Penrose, 1971]. Tensors are drawn as boxes, and summations over pairs of indices as wires. The "identity" tensor (i.e. 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:



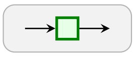
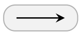
These diagrams are called tensor networks. We can then work directly with these graphs, expressing equations of tensor expressions as graph rewrites rules.



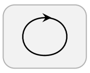
More generally, circuit diagrams, tensor networks, and many other graphical formalisms, can be expressed as arrows in some symmetric monoidal category. The diagrams above can then be interpreted as examples of a diagrammatic language common to all symmetric monoidal categories. These kinds of graphical languages introduce a particular challenge to formalising rewriting. For instance, consider a simple graph containing a self loop:

G : 

and a rewrite rule that rewrites the box to a line:

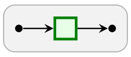
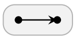
L :  $\Rightarrow$ R : 

Then, the graph resulting from rewriting the box with a self loop should be a circular edge with no vertices:



Graphs of this shape are beyond the normal notion of what one might consider 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, i.e. the dimension of the underlying vector space.

Suppose we tried naïvely to formalise this situation, by representing half-edges as edges connected to "dummy" points at the boundary.

L :  $\Rightarrow$ R : 

Then, 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 if 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 an graph with one input and one output, which contradicts the interpretation of rewrite rules 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 can address this problem uniformly by allowing edge-points. These extra "dummy" points 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 concerning 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 hyper-graphs 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 "graph" that allows for edges to be dangling at one or both ends, or be connected to themselves. We also consider these graphs as having a fixed interface, drawn as a collection of input and output wires and consider only 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 its reliance on adhesive categories [Lack and Sobocinski, 2005] and the *double-pushout* construction for graph rewriting [Ehrig et al., 1973]. 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.

Maps in many kinds of monoidal categories admit rich graphical languages [Selinger, 2009]. These languages become particularly interesting when one studies algebraic structures within monoidal categories. A developing field in category theory studies these algebras, and how they interact. [Lack, 2004] has shown that a certain class of these monoidal algebras, called *PROPs* can be composed in much the same way 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 [Bob Coecke, 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 this paper, we have clarified 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* of adhesive categories and inherit "enough adhesivity" to permit graph rewriting.

In particular, we introduce categories for open-graphs which are subcategories of slices over the category of graphs (**Graph**). 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*

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & B \arrow[d] \\
C \arrow[r] & D \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

Such that for any commutative cube

$$\begin{tikzcd}
& A' \arrow[rr] \arrow[dl] \arrow[dd] & & B' \arrow[dl] \arrow[dd] \\
C' \arrow[rr] \arrow[dd] & & D' \arrow[dd] & \\
& A \arrow[rr] \arrow[dl] & & B \arrow[dl] \\
C \arrow[rr] & & D &
\end{tikzcd}$$

where the back two faces are pullbacks, the following are equivalent:

- *the front two faces are pullbacks*
- *the top face is a pushout*

Definition 4.2. *[Lack and Sobocinski, 2005]. A category $\mathcal{A}$ is said to be* adhesive *if*

1. *$\mathcal{A}$ has pushouts along monomorphisms,*
2. *$\mathcal{A}$ has pullbacks,*
3. *and pushouts along monomorphisms in $\mathcal{A}$ are van Kampen squares.*

A crucial property of adhesive categories is that they have unique pushout complements over monomorphisms, when they exist.

Definition 4.3. *A* pushout complement *for a pair of arrows $(b : B \to K, f : K \to G)$, is another pair of arrows (c, g) such that*

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "c"'] & K \arrow[d, "f"] \\
G' \arrow[r, "g"'] & G \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

is a pushout.

Lemma 4.4. *[Lack and Sobocinski, 2005]. If a pair of arrows (b, f), where b is mono, has a pushout complement, it is unique up to isomorphism. That is, for any two pushout complements, (c, g) and (c', g'), there exists an isomorphism ϕ making the following diagram commute:*

$$\begin{tikzcd}
B \arrow[r, "c"] \arrow[d, "c'"'] & G' \arrow[d, "g"] \arrow[dl, "\phi"'] \\
G'' \arrow[r, "g'"'] & G
\end{tikzcd} \tag{2}$$

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 4.5 (Selective adhesive functor)**.** *Let $\mathcal{C}$ be a category and $\mathcal{A}$ be an adhesive category. A functor $S : \mathcal{C} \to \mathcal{A}$ is called a* selective adhesive functor *if it*

1. *is faithful,*
2. *preserves monomorphisms,*
3. *creates isomorphisms,*
4. *and reflects pushouts.*

Definition 4.6 (S-adhesive spans and pushouts)**.** *Let $S : \mathcal{C} \to \mathcal{A}$ be a selective adhesive functor. A span $A \xleftarrow{f} B \xrightarrow{g} C$ in $\mathcal{C}$ is called 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 4.7 (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.*

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "c"'] & K \arrow[d, "f"] \\
G' \arrow[r, "g"'] & G \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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 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 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 write simply $G -_f K$. 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 4.8. *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 $\mathcal{A}$.

$$\begin{tikzcd}
SB \arrow[r, "Sb"] \arrow[d, "Sc"'] & SK \arrow[d, "Sf"] \\
SG' \arrow[r, "Sg"'] & SG \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}
\qquad
\begin{tikzcd}
SB \arrow[r, "Sb"] \arrow[d, "Sc'"'] & SK \arrow[d, "Sf"] \\
SG'' \arrow[r, "Sg'"'] & SG \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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

$$\begin{tikzcd}
SB \arrow[r, "Sc"] \arrow[d, "Sc'"'] & SG' \arrow[d, "Sg"] \arrow[dl, "\phi'"'] \\
SG'' \arrow[r, "Sg'"'] & SG
\end{tikzcd}$$

Since S creates isomorphisms, there exists an iso $\phi : G' \to G''$ such that $S\phi = \phi'$. Substituting this map in, we have:

$$\begin{tikzcd}
SB \arrow[r, "Sc"] \arrow[d, "Sc'"'] & SG' \arrow[d, "Sg"] \arrow[dl, "S\phi"'] \\
SG'' \arrow[r, "Sg'"'] & SG
\end{tikzcd}$$

Diagram (2) commutes by the faithfulness of S. □

Definition 4.9 (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 the sake of conciseness, we will often denote a rewrite rule simply as $L \multimap R$, leaving the boundary maps implicit. When we do this, each time we write $L \multimap R$, it denotes the same rewrite rule, and in particular, it has the same boundary maps.

Definition 4.10 (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 4.11 (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:*

$$\begin{tikzcd}
L \arrow[d, "m"'] & B \arrow[l, "b_1"'] \arrow[r, "b_2"] \arrow[d, "c"] \arrow[dl, phantom, "\urcorner", very near end] \arrow[dr, phantom, "\ulcorner", very near end] & R \arrow[d] \\
G & G' \arrow[l] \arrow[r] & H
\end{tikzcd}$$

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 don't 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 4.12 (Rewrite system)**.** *A set of rewrite rules $\mathbb{S}$ is called a* rewrite system*. We define the relation $G \mathrel{-\!\!\rhd}_{\mathbb{S}} H$ 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 $\mathrel{-\!\!\rhd}_{\mathbb{S}}$ is denoted $\overset{*}{\mathrel{-\!\!\rhd}}_{\mathbb{S}}$, and the reflexive, symmetric, transitive closure as $\overset{*}{\mathrel{\lhd\!\!-\!\!\rhd}}_{\mathbb{S}}$.*

Theorem 4.13. *S-adhesive pushout complements commute with adhesive pushouts. Consider the following diagram, where b is mono, (b, m) has an S-adhesive pushout complement, and (p, q) and (p', q) are both S-adhesive spans.*

$$\begin{tikzcd}
B \arrow[r, "c"] \arrow[dd, "b"'] & G -_{b,m} K \arrow[dd, "s"'] & & \\
 & & P \arrow[ul, "p'"'] \arrow[dl, "p"] \arrow[r, "q"] & H \\
K \arrow[r, "m"'] & G \arrow[ul, phantom, "\ulcorner", very near start] & &
\end{tikzcd}$$

Then, for the pushout injections $i : G \hookrightarrow G +_{p,q} H$ and $i' : G -_{b,m} K \hookrightarrow (G -_{b,m} K) +_{p',q} H$, there is an open-graph isomorphism, commuting with the coboundaries c and c' of K in G and $G +_{p,q} H$ respectively.

$$\begin{tikzcd}
B \arrow[r, "c'"] \arrow[d, "c"'] & (G +_{p,q} H) -_{b,im} K \arrow[d, "\cong"] \\
G -_{b,m} K \arrow[r, "i'"] & (G -_{b,m} K) +_{p',q} H
\end{tikzcd} \tag{3}$$

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 $sp' = p$.

$$\begin{tikzcd}
& P \arrow[r, "q"] \arrow[d, "p'"] & H \arrow[dd] \\
B \arrow[r, "c"] \arrow[d, "b"] & G -_{b,m} K \arrow[d, "s"] & {[1]} \\
K \arrow[r, "m"'] & G \arrow[r, "i"'] \arrow[ul, phantom, "\ulcorner", very near start] & G +_{p,q} H \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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

$$\begin{tikzcd}
& P \arrow[r, "q"] \arrow[d, "p'"] & H \arrow[d] \\
B \arrow[r, "c"] \arrow[d, "b"] & G -_{b,m} K \arrow[r] \arrow[d, phantom, "{[2]}"] & (G -_{b,m} K) +_{p',q} H \arrow[d] \arrow[ul, phantom, "\ulcorner", very near start] \\
K \arrow[r, "m"'] & G \arrow[r, "i"'] & G +_{p,q} H \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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

$$\begin{tikzcd}
B \arrow[r] \arrow[d, "b"'] & (G +_{p,q} H) -_{b,im} K \arrow[d] \\
K \arrow[r, "im"'] & G +_{p,q} H \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

The result then follows from uniqueness of pushout complements. □

Theorem 4.14. *S-adhesive rewrites commute with S-adhesive pushouts. Let $m : L \to G$ be a matching of $L \multimap_{b_1,b_2} R$. The rewrite is computed as the double pushout:*

$$\begin{tikzcd}
L \arrow[d, "m"'] & B \arrow[l, "b_1"'] \arrow[r, "b_2"] \arrow[d, "c"] & R \arrow[d, "m'"] \\
G \arrow[ur, phantom, "\urcorner", very near start] & G -_{b,m} L \arrow[l, "s"] \arrow[r, "s'"'] & G[L \multimap R]_m \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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

$$\begin{tikzcd}
G & & \\
G -_{b_1,m} L \arrow[u, "s"] \arrow[d, "s'"'] & P \arrow[l, "p'"'] \arrow[r, "q"] \arrow[ul, bend right, "p"'] \arrow[dl, bend left, "\widehat{p}"] & H \\
G[L \multimap R]_m & &
\end{tikzcd} \tag{4}$$

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

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

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 Thm 4.13, 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.

$$\begin{tikzcd}
L \arrow[d, "im"'] & B \arrow[l] \arrow[r] \arrow[d, "c'"] & R \arrow[d] \\
G +_{p,q} H \arrow[ur, phantom, "\urcorner", very near start] & (G -_{b_1,m} L) +_{p',q} H \arrow[l] \arrow[r] & G[L \multimap R] +_{\widehat{p},q} H \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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

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, or 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 by which we connect open-graphs together. We prove several useful properties about the category **OGraph** of open-graphs, and show that the inclusion S : **OGraph** $\hookrightarrow$ **Graph**$/2_{\mathcal{G}}$ is a selective adhesive functor into the (adhesive) slice category **Graph**$/2_{\mathcal{G}}$.

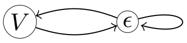
To fix notation, recall the standard definition for directed graphs as a functor category.

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

$$\begin{tikzcd}
E \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] & P
\end{tikzcd}$$

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 type graph $2_{\mathcal{G}}$ will be used to distinguish points that should be interpreted as "logical" vertices, from the "dummy"-points that occur along an edge:

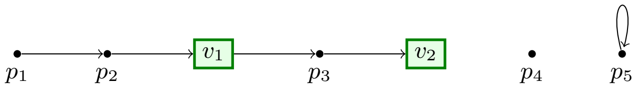
$2_{\mathcal{G}} :=$ 

A graph, G, is said to be typed by $2_{\mathcal{G}}$ when there is a typing morphism $\tau : G \to 2_{\mathcal{G}}$. When a vertex, $p \in G$, is mapped to V, i.e. $\tau(p) = V$, we refer to it as a *vertex-point* or simply as a *vertex.* The other points in G, those with $\tau(p) = \epsilon$, are called *edge-points.*

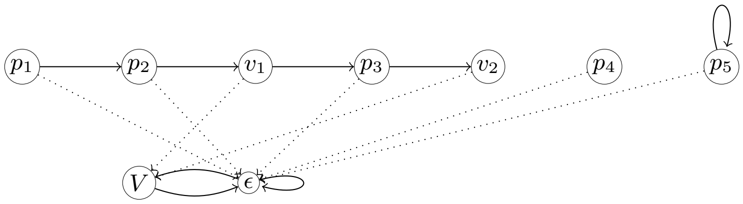
Definition 5.2 (**OGraph**)**.** *The category* ***OGraph*** *of* open-graphs*, is a subcategory of the slice category* ***Graph***$/2_{\mathcal{G}}$*. Objects are those of* ***Graph***$/2_{\mathcal{G}}$ *where each edge-point has at most one in-edge and one out-edge. The morphisms of* ***OGraph*** *are the same as those in* ***Graph***$/2_{\mathcal{G}}$*, with the additional restriction that they be full on vertices: any edge adjacent to a vertex $f(v)$ must also be in the image of f.*

This slice construction plays two roles. As well as distinguishing 'real' vertices from edge-points, 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. *A diagrammatic presentation of an open-graph:*



This diagram abbreviates a graph with its morphism to $2_{\mathcal{G}}$, which can otherwise be drawn in the more verbose fashion:



where the dotted arrows indicate the type-morphism for points, and the edge mapping is trivially inferred.

Definitions 5.4 (**OGraph** Notation)**.** *If an edge-point $p \in P_G$ has no in-edges, it is called an* input*. We write the set of inputs of G as $In(G)$. Similarly, an edge-point with no out-edges is called an* output*, and the set of outputs is written $Out(G)$. 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 of only isolated points is called a* point-graph*.*

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

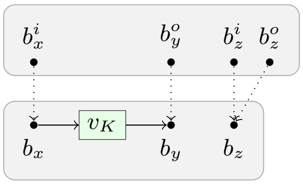
Definition 5.5 (Boundary Graph and Boundary Map)**.** *Given an open-graph G, its* boundary graph *is the point-graph formed from the coproduct of its inputs and outputs: $B := In(G) + Out(G)$. The* boundary map *of G is the induced map $b : B \to G$ of the inclusions of $In(G)$ and $Out(G)$.*

$$\begin{tikzcd}
In(G) \arrow[r, "b^i"] \arrow[dr, hook] & In(G) + Out(G) \cong B \arrow[d, dashed, "b"] & Out(G) \arrow[l, "b^o"'] \arrow[dl, hook'] \\
& G &
\end{tikzcd}$$

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

A boundary map identifies all the inputs and outputs of G, and is injective except on that isolated points of G, where it is 2-to-1.

Example 5.6. *The following illustrates a graph (below) with its boundary graph (above), where the boundary map is indicated by the dotted arrows.*



Notice that because each isolated-point is both an input and an output, a boundary graph has two points for each isolated point in its associated graph. It is important to note that a boundary map is mono if and only if its associated graph has no isolated points.

Definition 5.7 (Share the same boundary)**.** *Two graphs G and H are said to* share the same boundary*, B, by boundary maps b_1 and b_2, when $G \xleftarrow{b_1} B \xrightarrow{b_2} H$, $In(L) \cong In(R)$, $Out(L) \cong Out(R)$, and the following diagram commutes:*

$$\begin{tikzcd}
 & In(L) \arrow[rr, "\sim"] \arrow[dl, hook] \arrow[dr, "l^i"] & & In(R) \arrow[dl, "r^i"'] \arrow[dr, hook] & \\
L & & B \arrow[ll, "l"'] \arrow[rr, "r"] & & R \\
 & Out(L) \arrow[rr, "\sim"'] \arrow[ul, hook] \arrow[ur, "l^o"'] & & Out(R) \arrow[ul, "r^o"] \arrow[ur, hook] &
\end{tikzcd}$$

Notice that for two graphs with no isolated points, this condition means that boundaries of the two graphs are in bijection, and furthermore that bijection sends inputs to inputs and outputs to outputs.

We now show some basic properties of **OGraph**. In particular, we develop the properties needed to show that the inclusion of **OGraph** into $\mathbf{Graph}/2_{\mathcal{G}}$ is a selective adhesive functor.

Lemma 5.8. *A map in **OGraph** is a monomorphism iff it is injective.*

Proof. To prove the left to right direction of the iff, assume $f : G \to H$ is mono but not injective. Then there must be some point or edge, x in H, that has more than one point or edge in its pre-image. For K the smallest graph in G that contains a point or edge in the pre-image of x, there exist two distinct embeddings $i_1, i_2 : K \to G$ in $\mathbf{Graph}/2_{\mathcal{G}}$ such that $fi_1 = fi_2$. Either K is a single-vertex, an isolated point, or a single edge with two endpoints, thus it is an open-graph, but i_1 and i_2 are not necessarily full on vertices. However, the induced maps $[1_G, i_1] : G + K \to G$ and $[1_G, i_2] : G + K \to G$ are still distinct *and* are full on vertices, where $+$ is the coproduct of $\mathbf{Graph}/2_{\mathcal{G}}$ and 1_G is the identity on G. Thus:

$$f \circ [1_G, i_1] = [f, fi_1] = [f, fi_2] = f \circ [1_G, i_2]$$

This implies that f is not mono in **OGraph**, and from this contradiction we get that f is injective.

The reverse direction of the iff follows from injective morphisms in **Graph** being monos. □

We will now prove a similar result for surjections. However, **OGraph** is quite restrictive on the types of maps and graphs that exist, so not all epimorphisms are surjective. However, all *strong* epimorphisms are. To show this, we first recall the notion of strong epimorphism and prove a simple fact about surjections.

Definition 5.9. *A* strong epimorphism *in $\mathcal{C}$ is an epimorphism e that is left-orthogonal to all monomorphisms in $\mathcal{C}$. That is, for any commutative square of the following form, with m as a monomorphism:*

$$\begin{tikzcd}
A \arrow[r, two heads, "e"] \arrow[d, "f"'] & B \arrow[d, "g"] \arrow[dl, dashed, "d"'] \\
C \arrow[r, hook, "m"'] & D
\end{tikzcd}$$

there exists a unique diagonal map, d, making the diagram commute.

Lemma 5.10. *For the following commutative triangle:*

$$\begin{tikzcd}
A \arrow[r, "e"] \arrow[dr, "f"'] & B \arrow[d, "f'"] \\
 & C
\end{tikzcd}$$

where e is a surjection, f is full on vertices iff f' is.

Proof. First, consider the case when f' is full on vertices. All surjections are full on vertices, thus $f'e$ is also full on vertices. Now, suppose f is full on vertices. Note that precomposing with a surjection does not affect the image of a map, so $f[A] = f'e[A] = f'[B]$. So f' is full on vertices. □

Lemma 5.11. *A map in **OGraph** is a strong epimorphism iff it is surjective.*

Proof. We first show that surjections in **OGraph** are strong epimorphisms. First note that surjections in **OGraph** are strong epimorphisms in **Graph**/$2_{\mathcal{G}}$; monos in **OGraph** are injections and hence monos in **Graph**/$2_{\mathcal{G}}$. Therefore, it suffices to show that the diagonal map d from Def 5.9 is full on vertices. This follows from Lem 5.10.

To show that all strong epimorphisms are surjections, we begin by noting that strong epimorphisms are, in particular, extremal epimorphisms. That is, given an epimorphism e, such that for any factorisation $e = mf$, where m is mono, then m must be an isomorphism. Suppose some map e does not have this property, then it factors as $e = mf$ for some monomorphism that is not an isomorphism. Then m must not be surjective, so e must also not be. From this contradiction, strong epimorphisms in **OGraph** are all surjections. □

Lemma 5.12. ***OGraph*** *has unique strong epi-mono factorisations.*

Proof. If a map f factors as me, where e is a strong epimorphisms and m is a monomorphism, this factorisation is automatically unique. This factorisation exists because any map f factors through its image:

$$A \overset{f_e}{\twoheadrightarrow} f[A] \overset{f_m}{\hookrightarrow} B$$

The map f_e of f onto its image is full on vertices because it is surjective, and the embedding f_m of the image of f in B is full on vertices precisely when f is. By Lemmas 5.8 and 5.11, all strong epi-mono factorisations are of this form. □

Lemma 5.13. *The embedding functor S :* ***OGraph*** *$\hookrightarrow$* ***Graph***/$2_{\mathcal{G}}$ *preserves and reflects monomorphisms and strong epimorphisms, and it creates isomorphisms.*

Proof. Monos and strong epimorphisms follow from Lemmas 5.8 and 5.11. Creation of isomorphisms follows from the fact that the definition of open-graph is invariant under isomorphism and all isomorphisms are full on vertices. □

Lemma 5.14. *The embedding functor S reflects colimits.*

Proof. Let the following diagram be a coequaliser in **Graph**/$2_{\mathcal{G}}$:

$$\begin{tikzcd}
SA \arrow[r, shift left, "Sf"] \arrow[r, shift right, "Sg"'] & SB \arrow[r, "Sq"] & SQ
\end{tikzcd}$$

Then, because S is faithful, $qf = qg$. Suppose there is some q' in **OGraph** such that $q'f = q'g$, then there exists unique u in **Graph**/$2_{\mathcal{G}}$ making this diagram commute:

$$\begin{tikzcd}
SA \arrow[r, shift left, "Sf"] \arrow[r, shift right, "Sg"'] & SB \arrow[r, "Sq"] \arrow[dr, "Sq'"'] & SQ \arrow[d, dashed, "u"] \\
 & & SQ'
\end{tikzcd}$$

Sq is a regular epimorphism in **Graph**/$2_{\mathcal{G}}$, so in particular it is a strong epimorphism. By Lem 5.13, q is a strong epimorphism in **OGraph**. Thus, by Lemma 5.10 u is full on vertices, or equivalently, $u = Su'$ for some (unique) u' in **OGraph**. So, by faithfulness of S, the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B \arrow[r, "q"] \arrow[dr, "q'"'] & Q \arrow[d, dashed, "u'"] \\
 & & Q'
\end{tikzcd}$$

Let SC be a set-indexed coproduct in **Graph**/$2_{\mathcal{G}}$, with injections $Si_j : SG_j \to SC$. Then, for any arrows $f_j : G_j \to H$, there exists a unique u making the following diagram commute for all j:

$$\begin{tikzcd}
SG_j \arrow[r, "Si_j"] \arrow[dr, "Sf_j"'] & SC \arrow[d, dashed, "u"] \\
& SH
\end{tikzcd}$$

The image of u is the union of the images of all the maps f_j, so it is full on vertices. Therefore $C \cong \coprod G_j$ is a coproduct in **OGraph**. □

Theorem 5.15. *The embedding functor S : **OGraph** $\hookrightarrow$ **Graph**$/2_{\mathcal{G}}$ is a selective adhesive functor.*

6 Composition and Decomposition for Open Graphs

In this section, we show how open-graphs can be composed and decomposed using the embedding functor **OGraph** $\to$ **Graph**$/2_{\mathcal{G}}$ which we call S. In particular, using the fact that S is a selective adhesive functor, we introduce definitions for subtracting one graph from another (by pushout complements) and for connecting open-graphs along their boundary.

We will first define the spans of graphs that preserve open-graphs under pushout. The crucial features of such pushouts are:

- if two edges are identified, then the whole paths of edge-points they are on are also identified: never identify only a middle section of one edge with a middle section of another edge;
- outputs should only be connected to inputs: never connect the output of an edge to the output of another edge, and likewise with inputs.

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

Definition 6.1 (Boundary Coherent)**.** *A span $H_1 \xleftarrow{f} G \xrightarrow{g} H_2$ is called* boundary coherent *when f and g are monos and:*

1. *for all $p \in In(G)$ at most one of $f(p)$ and $g(p)$ is an input;*
2. *for all $p \in Out(G)$ at most one of $f(p)$ and $g(p)$ is an output.*

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

Theorem 6.2. *Boundary-coherent spans are S-adhesive spans.*

Proof. Let $A \xleftarrow{f} B \xrightarrow{g} C$ be a boundary coherent span. Then the following is a pushout in **Graph**$/2_{\mathcal{G}}$.

$$\begin{tikzcd}
SA \arrow[r, "Sf"] \arrow[d, "Sg"'] & SB \arrow[d, "p_1"] \\
SC \arrow[r, "p_2"'] & D \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

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**$/2_{\mathcal{G}}$ 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.

$$\begin{tikzcd}
SB \cap SC \arrow[r, hook, "Sf"] \arrow[d, hook, "Sg"'] & SB \arrow[d, hook, "i_1"] \\
SC \arrow[r, hook, "i_2"'] & SB \cup SC \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}$$

Suppose an edge point p in $SB \cup SC$ has two out-edges. Then one must be in SB and the other in SC. Thus neither are in the intersection, so 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. Moreover, i_1 and i_2 are full on vertices because Sf and Sg are, so (Sf, Sg) is an S-adhesive span. □

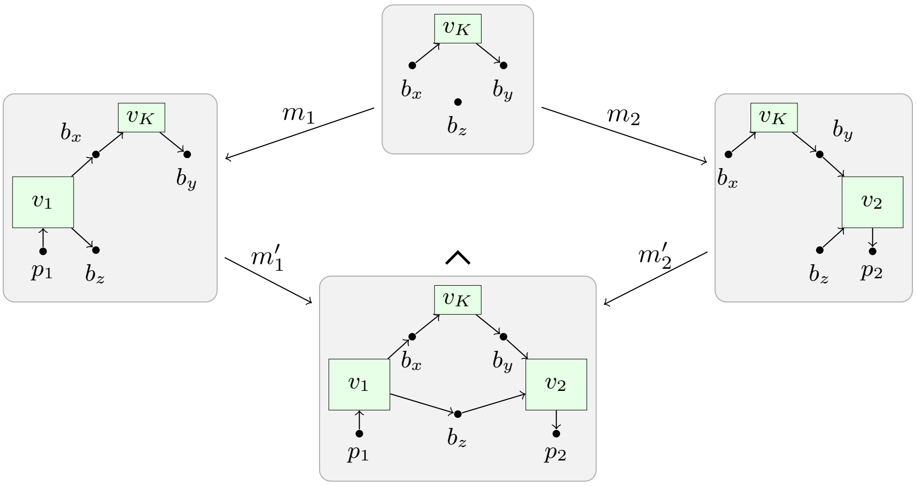
This provides boundary-coherent spans with nice properties for pushouts as reflected by the selective adhesive functor S. These pushouts, which we call *mergings*, will play a central role in our construction of rewriting.

Definition 6.3 (Merging)**.** *Given a boundary-coherent span of monos $G_1 \overset{m_1}{\hookleftarrow} K \overset{m_2}{\hookrightarrow} G_2$, 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:*

$$\begin{tikzcd}
& K \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
G_1 \arrow[dr, "m_1'"'] & \wedge & G_2 \arrow[dl, "m_2'"] \\
& M &
\end{tikzcd}$$

This makes M the smallest graph containing G_1 and G_2 with a single copy of the shared sub-graph K, as identified by m_1 and m_2.

Example 6.4. *An illustration of merging graphs:*

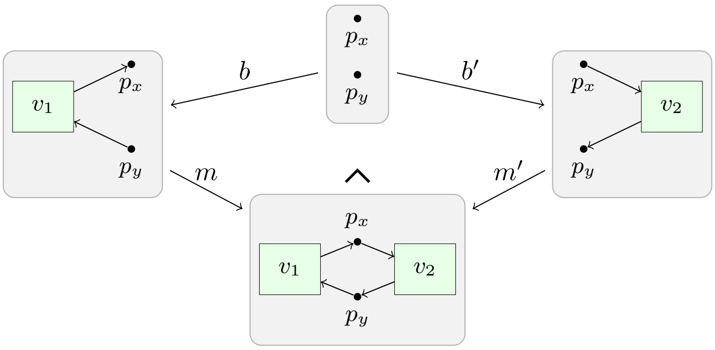


The grey boxes are drawn around the graphs involved to distinguish between edges in the graphs and those of the pushout diagram. The image of the maps are indicated by the naming of edge-points and vertices.

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

Definition 6.5 (Plugging)**.** *A 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 graph being merged on is a point graph; i.e. in the span $G_1 \overset{b_1}{\hookleftarrow} P \overset{b_2}{\hookrightarrow} G_2$, P is a point-graph.*

Example 6.6. *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 graphs side by side.

We now introduce a dual notion to merging, called *subtraction* which is formed by S-adhesive pushout complements. Intuitively subtraction removes part of an open-graph identified by a monomorphism. We first give a concrete definition for subtraction and then we show that this definition does indeed produce S-adhesive pushout complements.

Definition 6.7 (Subtraction)**.** *We define the* subtraction *of G from M, at a mono $m : G \hookrightarrow M$, written $M -_m G$, as the graph H defined by:*

$$\begin{aligned} P_H &= (P_M \backslash m[P_A]) + P_B \\ E_H &= (P_M \backslash m[P_A]) \end{aligned}$$

This removes all of G from M, but re-introduces the vertices from the boundary graph B of G. Let $b : B \to G$ be the boundary map of G. The source and target maps are then defined as follows for each $e \in E_H$, which includes edges that formerly went to boundary points in G:

$$s_H(e) = \begin{cases} b^o(p) & \textit{if } p \in \mathit{Out}(G) \textit{ and } m(p) = s_M(e) \\ s_M(e) & \textit{otherwise} \end{cases}$$

$$t_H(e) = \begin{cases} b^i(p) & \textit{if } p \in \mathit{In}(G) \textit{ and } m(p) = t_M(e) \\ t_M(e) & \textit{otherwise} \end{cases}$$

The process of removing the image of m leaves some edges without a source or a target. In that case, the edge is re-connected to a point in P_B. We call the induced embedding, $c : B \hookrightarrow H$, the coboundary *of b with respect to m. When the embedding of G into M is implicit, we 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 and well-defined.

Proof. The source map s_H is total because the source of an edge, $e \in E_H$, is in the image of m iff it is an output of G. Because m is mono, s_H is well-defined. Similarly for t_H. □

Theorem 6.8. *Subtractions by graphs without isolated points are S-adhesive pushout complements: given $H := M -_m G$, the following diagram is an S-adhesive pushout:*

$$\begin{tikzcd} B \arrow[r, "b"] \arrow[d, "c"'] & G \arrow[d, "m"] \\ H \arrow[r, "f"'] & M \arrow[ul, phantom, "\ulcorner", very near start] \end{tikzcd} \tag{5}$$

where b is the boundary map of G, and c is the coboundary of m.

Proof. First, we show b, c is boundary coherent. 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, the resulting graph M'' is isomorphic to M, and for the induced embedding m'' of G into M'', the following diagram commutes:

$$\begin{tikzcd} G \arrow[r, "m"] \arrow[d, "m''"'] & M \\ M'' \arrow[ur, "\cong"'] & \end{tikzcd}$$

So, diagram (5) is also a pushout square for some f. □

Since G contains no isolated points, the maps b and c from the theorem are mono. As B is a point graph, this implies (b, c) defines a plugging, and $H := M -_m G$ is the uniquely determined open-graph such that $G +^*_{b,c} H \cong M$.

7 Rewriting with Open-Graphs

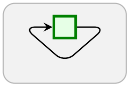
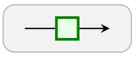
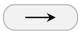
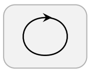
We now introduce rewrite rules for open-graphs, how they can be applied, under what conditions they can commute with merging and plugging, and how rewrites can themselves be composed. Finally we present a rewrite system called edge-homeomorphism that lets us ignore intermediate edge-points.

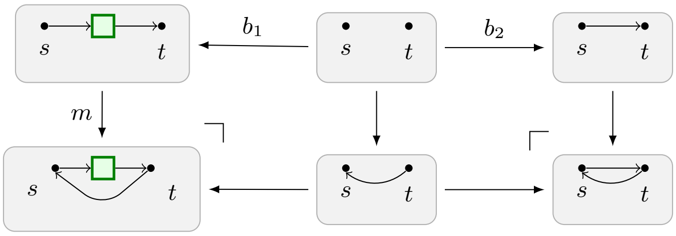
Definition 7.1 (Rewrite)**.** *A span $L \xleftarrow{b_1} B \xrightarrow{b_2} R$, in which L and R share the same boundary, B, by monos b_1 and b_2, is called a* rewrite rule *and is written $L \multimap_{b_1,b_2} R$. The rewrite rule is said to* rewrite G to G' *at a mono $m : L \hookrightarrow G$,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

$$\begin{tikzcd}
L \arrow[d, hook, "m"'] & B \arrow[l, hook', "b_1"'] \arrow[r, hook, "b_2"] \arrow[d, hook] & R \arrow[d, hook] \\
G \arrow[ur, phantom, "\lrcorner", very near start] & G -_m L \arrow[l, hook'] \arrow[r, hook] & G' \arrow[ul, phantom, "\llcorner", very near start]
\end{tikzcd}$$

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

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

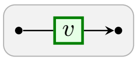
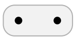
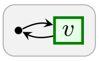
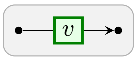
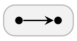
Example 7.2 (Circles)**.** *We now return to the challenging example introduced at the end of §2. We will rewrite the graph*  *by*  $\multimap$  *to get*  *. The pushout construction for this rewrite is as follows:*



Notice that the above rewrite contains additional intermediate edge-points. Informally, these are intended to be treated as part of the edge. In §7.3, we formalise this idea by introducing rewrite rules that insert and 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 $G :=$* *, $H :=$* *. For $K :=$* *, we can find maps $f : K \to G, g : K \to H$ such that the S-adhesive pushout $G +_{f,g} H :=$* *. While the left-hand side of the rewrite*  $\multimap$  *matches H, it does not match $G +_{f,g} H$.*

We will be primarily concerned with pluggings, and so we now provide a precise definition of what it means for a plugging and a rewrite to be compatible.

Definition 7.4 (Compatible)**.** *A plugging $G +^*_{p,q} H$ and a rewrite $G[L \multimap R]_m$ 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:*

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

We can actually show that *all* pluggings and rewrites are compatible. Before we prove this important theorem, we first show that the boundary of an open-graph in 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 $G[L \multimap R]_m$:*

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, very near start, "\urcorner"] & B \arrow[l, "b_1"'] \arrow[r, "b_2"] \arrow[d, "c"] & R \arrow[d, "m'"] \arrow[dl, phantom, very near start, "\ulcorner"] \\
G & G -_m L \arrow[l, "s"'] \arrow[r, "s'"] & G[L \multimap R]_m \\
 & B' \arrow[ul, "b_1'"] \arrow[u, "k"'] \arrow[ur, "b_2'"'] &
\end{tikzcd}$$

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 can show that the boundary of G is in the image of s, by the definition of subtraction. If some point $x\ B'$ 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 $G -_m L$, x must be in the image of s. Thus, for all $x\ B'$, there exists x' in $G -_m L$ such that $s(x') = x$. Since s is mono, x' is unique, so let k be defined as the map sending x to x', and let $b_2' = s'k$.

It suffices to show that $s'k$ is a boundary map. If x is an input of G, then $k(x)$ is either still an input or becomes 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 the combined graph. This follows similarly for outputs. It can also be shown that $s'k$ covers the boundary of $G[L \multimap R]_m$, so it is a boundary map. □

We can now use this theorem and Thm 4.14 to show not only that pluggings and rewrites are compatible, but explicitly define the maps $\widehat{m}$ and $\widehat{p}$ used in Def 7.4.

Theorem 7.6. *Rewriting and plugging are compatible. Suppose $(p : K \to G, q : K \to H)$ is a plugging and i the embedding of G into $G +^*_{p,q} H$. Let $m : L \to G$ be a matching of a rewrite rule $L \multimap R$. 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 p is mono, i is mono, 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 Thm 7.5, let $p' = kr$ and $\widehat{p} = b_2' r$. Then by the above theorem, the following diagram commutes:

$$\begin{tikzcd}
G & & \\
G -_m L \arrow[u, "s"] \arrow[d, "s'"'] & P \arrow[ul, bend right, "p"'] \arrow[l, "p'"'] \arrow[r, "q"] \arrow[dl, bend left, "\widehat{p}"] & H \\
G[L \multimap R]_m & &
\end{tikzcd}$$

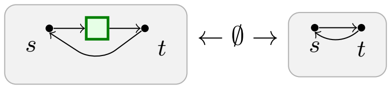
If $p(x)$ is an input, then $p'(x)$ and $\widehat{p}(x)$ are both inputs, and similarly for outputs. Therefore (p, q), (p', q), and $(\widehat{p}, q)$ are all boundary-coherent spans, and hence S-adhesive spans. The result then follows from Thm 4.14. □

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 $G[L \multimap R]_m$, then the rewrite rule $G \multimap_{b_1,b_2} G[L \multimap R]_m$ is called the* extension *of r by m, and written $r^{\uparrow m}$.*

Notice that this is a well defined rewrite rule because the boundary span b_1, b_2 is uniquely defined by Thm 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 $\emptyset$.

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

We now show how rules can be directly combined using the underlying operations on open-graphs. 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 $r_1 := L_1 \multimap R_1$ and $r_2 := L_2 \multimap R_2$ and a merged graph $M := R_1 +_{k_1,k_2} L_2$, then 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 Thm 7.5:

$$\begin{tikzcd}
M[R_1 \multimap L_1]_{m_1} & \cdot \arrow[l] \arrow[r] & M & \cdot \arrow[l] \arrow[r] & M[L_2 \multimap R_2]_{m_2} \\
 & & B \arrow[ull, "b_1"] \arrow[ul] \arrow[u] \arrow[ur] \arrow[urr, "b_2"'] & &
\end{tikzcd}$$

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 §2. Sequential composition is correct in the sense that it does nothing more than $\overset{*}{\rightarrowtriangle}_{\mathbb{S}}$.

Theorem 7.10 (Soundness)**.** *if $(r_1 \,;_{k_1,k_2} r_2) := G \multimap G'$ is a rewrite; then there exists a graph H, and monos m_1 and m_2 such that $G \overset{r_1{}^{\uparrow m_1}}{\multimap} M \overset{r_2{}^{\uparrow m_2}}{\multimap} G'$.*

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. Thus what we have left to prove is 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. □

Sequential composition of rewrites is also complete in the sense that many rewrites under various extensions can also be represented as the sequential composition of the rewrites under a single extension.

Theorem 7.11 (Completeness)**.** *if $M_1 \overset{r_1{}^{\uparrow m_1}}{\multimap} M_2 \overset{r_2{}^{\uparrow m_2}}{\multimap} M_3$ then 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 a graph K which 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 $L_2' := L_2 - K$ and $M_2' := (M_2 - R_1) - L_2'$. 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. □

7.3 Edge-Homeomorphism

Although we have defined everything discreetly 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.

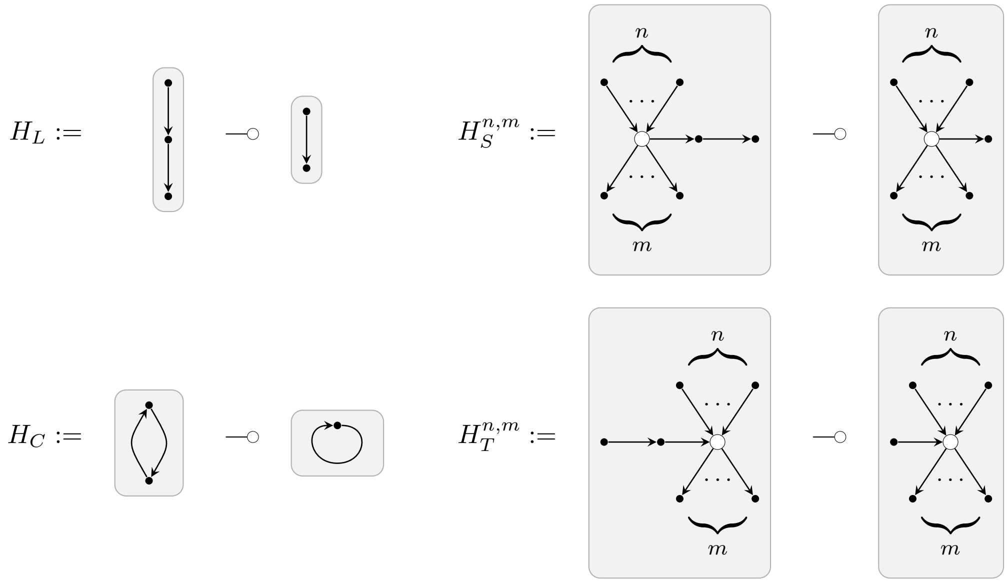
Definition 7.12. *For an open-graph G, a* wire W *in G is a set of connected edge-points, which contains at least one edge, and may also include vertices at its start and end. If a vertex is connected to either end of W in G, it is called an* endpoint *of W.*

As 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 where $t(e_1) = s(e_2)$, 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 graphs G and G' are called* 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 is called* edge-homeomorphism *and denoted by $\mathbb{H}$:*



Applying edge-homeomorphism rewrites to a graph, from left to right, is called *contracting*. 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. If G rewrites to H using zero or more edge homeomorphism rewrites, we say H is an *edge contraction* of G.

Lemma 7.15. *The rewrite system $\mathbb{H}$ is confluent and terminating.*

Proof. Termination comes from observing that each contraction of a morphism decreases the number of edge-points. Confluence comes from observing that any two contractions result in isomorphic graphs independently of the order they are applied (there are no critical pairs). □

Considering 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

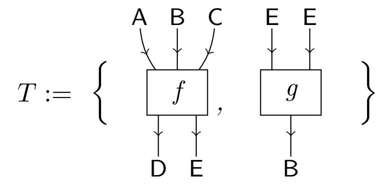
We now generalise our definition of open-graph by showing how it can be parametrised by a 'graphical signature' to form a notion of typed open-graphs. The graphical signature defines the types and arities of vertices, as well as the types of edges which can be used. This generalised construction makes 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 rewriting properties are inherited in typed open-graphs.

Definition 8.1. *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*

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

can be visualised as a set of "boxes":

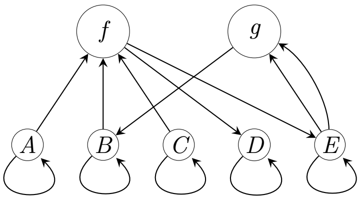


(6)

Remark 8.3. *Graphical signatures are essentially what Selinger calls a* monoidal signature *[Selinger, 2009] and Joyal and Street call a* tensor scheme *[Joyal and Street, 1991]. We shall see in §9 the relationship between these maps and the construction of free monoidal categories.*

For a graphical signature T, we can form a *typegraph* $T_{\mathcal{G}}$ as follows. It has as vertices $O + A$, where every $o \in O$ has a self-loop. For $a \in A$, $T(a)$ is a pair of words D, C, defining the domain and codomain of a; in particular defining the types of the inputs and outputs of a respectively. For each d in D, $T_{\mathcal{G}}$ has an edge from d to a. For each c in C, $T_{\mathcal{G}}$ has an edge from a to c. Note that the in-edges and out-edges of each vertex in $T_{\mathcal{G}}$ have a natural total order given by their word order.

Example 8.4. *T defined as in (6) defines the typegraph $T_{\mathcal{G}}$:*



Definitions 8.5 (Typegraph Notation)**.** *For a $T_{\mathcal{G}}$-graph (G, τ), points $p \in \tau^{-1}(O)$ are called* edge-points. *All other points are called* vertices.

For such graphs, we want to have a property even stronger than fullness on vertices. Whereas in the previous section, adjacent edges of $f(v)$ only had to be *covered* by f, here they must be in 1-to-1 correspondence with the adjacent edges of v. For some vertex v in G, we call the set of adjacent edges $N(v)$ its *edge neighbourhood*, and define local isomorphism as follows:

Definition 8.6 (Local Isomorphism)**.** *A map $f : G \to H$ is called a* local isomorphism*, for every vertex $v \in G$, the edge function of f restricts to an bijection $f^v : N(v) \xrightarrow{\sim} N(f(v))$.*

In particular, we can regard the type map $\tau : G \to T_{\mathcal{G}}$ as an arrow from τ to $1_{T_{\mathcal{G}}}$ in the slice category **Graph**$/T_{\mathcal{G}}$, and ask that it be a local isomorphism.

Let $(\mathbf{Graph}/T_{\mathcal{G}})_{\cong}$ be the subcategory of **Graph**$/T_{\mathcal{G}}$ whose objects are pairs $(G, \tau : G \to T_{\mathcal{G}})$ where τ is a local isomorphism, and whose arrows are local isomorphisms. We can show this subcategory is in fact full.

Lemma 8.7. $(\boldsymbol{Graph}/T_{\mathcal{G}})_{\cong}$ *is a full subcategory of* $\boldsymbol{Graph}/T_{\mathcal{G}}$.

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

$$\begin{tikzcd}
G \arrow[r, "\tau_G"] \arrow[d, "f"'] & T_{\mathcal{G}} \\
H \arrow[ur, "\tau_H"'] &
\end{tikzcd}$$

Thus, for any v in G we get this triangle in **Set**:

$$\begin{tikzcd}
N(v) \arrow[r, "\tau_G^v"] \arrow[d, "f^v"'] & N(\tau_G(v)) \\
N(f(v)) \arrow[ur, "\tau_H^{f(v)}"'] &
\end{tikzcd}$$

Since τ_G^v and $\tau_H^{f(v)}$ are both bijections, f^v is a bijection, so $(\mathbf{Graph}/T_{\mathcal{G}})_{\cong}$ is a full subcategory. □

Note that for any $T_{\mathcal{G}}$, there is a graph homomorphism $\kappa : T_{\mathcal{G}} \to 2_{\mathcal{G}}$ sending every point in O to ϵ and every point in A to V. Post-composing each object in $\mathbf{Graph}/T_{\mathcal{G}}$ with κ yields the forgetful functor:

$$U_\kappa : \mathbf{Graph}/T_{\mathcal{G}} \to \mathbf{Graph}/2_{\mathcal{G}}.$$

In particular, this sends an object $\tau : G \to T_{\mathcal{G}}$ in $\mathbf{Graph}/T_{\mathcal{G}}$ to an object $\kappa \circ \tau$ in $\mathbf{Graph}/2_{\mathcal{G}}$.

Definition 8.8 (Open $T_{\mathcal{G}}$-graph)**.** *A $T_{\mathcal{G}}$-graph G is called an open $T_{\mathcal{G}}$-graph if $U_\kappa(G) \in$ **Graph**$/2_{\mathcal{G}}$ is an open-graph. The category **OGraph**$_{T_{\mathcal{G}}}$ the full subcategory of (**Graph**$/T_{\mathcal{G}})_{\cong}$ whose objects are open-graphs.*

Note that local isomorphisms are, in particular, full on vertices, so the forgetful functor U_κ restricts to another functor

$$U : \mathbf{OGraph}_{T_{\mathcal{G}}} \to \mathbf{OGraph}.$$

Lemma 8.9. *Monos in **OGraph**$_{T_{\mathcal{G}}}$ are injective maps.*

Proof. Suppose $m : G \to H$ in $\mathbf{OGraph}_{T_{\mathcal{G}}}$ is not injective. If m takes two distinct edges e_1 and e_2 to a single edge, then suppose the source of e_1 (and hence of e_2) is an edge-point. Then, since G is an open-graph, it must take two edge-points to a single edge-point in H. Otherwise, suppose it is a vertex, then by local isomorphism, m must take two distinct vertices on to a single vertex. Thus is 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 graph consisting of a single edge-point. In either case, there are at least two distinct maps $f, g : K \to G$ such that $mf = mg$. □

Theorem 8.10. *The embedding functor $S' :$ **OGraph**$_{T_{\mathcal{G}}} \hookrightarrow$ **Graph**$/T_{\mathcal{G}}$ is a selective adhesive functor.*

Proof. From Lem 8.9, S' preserves monos. Creation of isomorphisms follows from the fact that all isomorphisms are local isomorphisms and the property of being an open-graph is invariant under isomorphism. Faithfulness and reflection of pushouts follows from being a full subcategory embedding. □

Definition 8.11 (Boundary-coherence in $\mathbf{OGraph}_{T_{\mathcal{G}}}$)**.** *A span $A \xleftarrow{f} B \xrightarrow{g} C$ in **OGraph**$_{T_{\mathcal{G}}}$ is called* boundary-coherent *if its image under U is boundary-coherent in **OGraph**.*

Theorem 8.12. *Boundary-coherent spans in **OGraph**$_{T_{\mathcal{G}}}$ are S' adhesive.*

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

$$\begin{tikzcd}
\mathbf{OGraph}_{T_{\mathcal{G}}} \arrow[r, "S'"] \arrow[d, "U"'] & \mathbf{Graph}/T_{\mathcal{G}} \arrow[d, "U_\kappa"] \\
\mathbf{OGraph} \arrow[r, "S"] & \mathbf{Graph}/2_{\mathcal{G}}
\end{tikzcd}$$

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

$$\begin{tikzcd}
S'(A) \arrow[r, "S'(f)"] \arrow[d, "S'(g)"'] & S'(B) \arrow[d, "p_2"] \\
S'(C) \arrow[r, "p_1"] & D
\end{tikzcd}$$

Since $\mathbf{OGraph}_{T_\mathcal{G}}$ is a full subcategory of $\mathbf{Graph}/T_\mathcal{G}$, it suffices to show that D is in $\mathbf{OGraph}_{T_\mathcal{G}}$. By definition, $U(f), U(g)$ is boundary-coherent and hence S-adhesive in **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 RHS, so by uniqueness of pushouts, $S(D') \cong U_\kappa(D)$. D' is an open-graph in **OGraph**, so D is an open-graph in $\mathbf{OGraph}_{T_\mathcal{G}}$. □

Boundary maps are defined as in **OGraph**. The construction of subtraction carries over verbatim, and is preserved by U. The uniqueness of pushout complements follows from adhesiveness of $\mathbf{Graph}/T_\mathcal{G}$.

9 Monoidal Theories

Plugging gives us a tool for composing graphs. We can take this a step further and discuss composing graphs in a *categorical* sense, using cospan categories over **OGraph** or $\mathbf{OGraph}_{T_\mathcal{G}}$. For our purposes, we shall focus on the latter.

For a graphical signature $T : A \to O^* \times O^*$, we construct the category $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ of *directed cospans* as follows. Its objects are words in O^*. Equivalently, they are point graphs in $\mathbf{OGraph}_{T_\mathcal{G}}$, where the points are given a total order. An arrow $G : X \to Y$ is a cospan

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

where G doesn't contain any isolated points, d is the inclusion of $\mathrm{In}(G) \cong X$, and c is the inclusion of $\mathrm{Out}(G) \cong Y$.

$\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ forms a symmetric monoidal category. Composition of maps $G : A \to B$ and $H : B \to C$ is by pushout, which is boundary-coherent by construction. For a point graph A, the identity of A in $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ is the cospan given by the identity of A in $\mathbf{OGraph}_{T_\mathcal{G}}$.

$$A \xrightarrow{1} A \xleftarrow{1} A$$

The monoidal product is given by coproducts in $\mathbf{OGraph}_{T_\mathcal{G}}$. For cospans $G : A \to B$, $H : C \to D$, $G \otimes H$ is the cospan

$$B + D \xrightarrow{o} G + H \xleftarrow{i} A + C,$$

where i and o are the induced maps of coproducts.

Symmetries $\sigma_{A,B} : A \otimes B \to B \otimes A$ are built using the induced swap map $\sigma := [i_2, i_1]$, for i_1 and i_2 the coproduct injections of $A + B$.

$$B + A \xrightarrow{1} B + A \xleftarrow{\sigma} A + B$$

Remark 9.1. *$\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ is actually a monoidal 2-category, where composition and the monoidal product are only associative up to isomorphism. It has as objects point-graphs, as 1-cells cospans, and as 2-cells $T_\mathcal{G}$-graph morphisms. For our purposes, we will work with the "strictified" category, where composition and $\otimes$ are both taken to be strictly associative. By a minor abuse of notion, for cospans $\mathcal{G}$, $\mathcal{H}$, $\mathcal{G} \cong \mathcal{H}$ should be read as a 2-cell isomorphism in the (non-strict) 2-category.*

9.1 Rewrite Categories

Lemma 9.2. *Let the following cospan be an arrow in $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$.*

$$\mathcal{G} := Y \xrightarrow{c} G \xleftarrow{d} X$$

Let m be a matching of a rewrite $L \multimap R$ on G. Then for the induced rewrite

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

there exists unique $\widehat{d}$, $\widehat{c}$ such that

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

is an arrow in $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ and the following diagram commutes for some maps d' and c'.

$$\begin{tikzcd}
 & G & \\
Y \arrow[ur, "c"] \arrow[r, "c'"] \arrow[dr, "\widehat{c}"'] & B \arrow[u, "b_1"'] \arrow[d, "b_2"] & X \arrow[ul, "d"'] \arrow[l, "d'"'] \arrow[dl, "\widehat{d}"] \\
 & G[L \multimap R]_m &
\end{tikzcd}$$

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

$$G \xleftarrow{b'_1} \mathrm{In}(G) \cong \mathrm{In}(G[L \multimap R]_m) \xrightarrow{b'_2} G[L \multimap R]_m$$

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 for c similarly. □

Definition 9.3 (Rewriting on Cospans)**.** *For a cospan*

$$\mathcal{G} := Y \xrightarrow{c} G \xleftarrow{d} X$$

in ***DCsp***(***OGraph***$_{T_\mathcal{G}}$)*, and a matching m of a rewrite $L \multimap R$ on G, we write $\mathcal{G}[L \multimap R]_m$ for the cospan over $G[L \multimap R]_m$ defined by Lem 9.2.*

Theorem 9.4. *Let $\mathcal{G} : A \to B$, $\mathcal{H} : B \to C$ be cospans in* ***DCsp***(***OGraph***$_{T_\mathcal{G}}$)*, and m be a matching of a rule $L \multimap R$ on G. Then there exists a matching m' on $\mathcal{H} \circ \mathcal{G}$ such that*

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

Similarly, for any cospan matching n on $\mathcal{H}$, there exists n' such that

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

Proof. The result follows from Thm 7.6 and Lem 9.2. In both cases, m' and n' are formed by composing the original mapping with the inclusion of the matched graph into the (S'-adhesive) pushout. □

Let $\mathbb{S}$ be a set of rewrite rules. We write $\mathcal{G} \rightarrowtriangle \mathcal{H}$ if there exists a rule $L \multimap R$ in $\mathbb{S}$ and a cospan matching m such that $\mathcal{G}[L \multimap R]_m \cong \mathcal{H}$. Let $\overset{*}{\leftrightarrowtriangle}$ be the closure of $\rightarrowtriangle$ as an equivalence relation.

Let $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})/\!\!/\, \mathbb{S}$ be the category whose objects are the same as those of $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ and whose arrows are equivalence classes of cospans under the relation $\overset{*}{\leftrightarrowtriangle}$. This category is well-defined because of Thm 9.4, and inherits its symmetric monoidal structure from $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$.

Let $\mathbb{H}$ be the typed version of the edge homeomorphism rewrite system from Def 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.

A particularly important example of a rewrite category is then $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})/\!\!/\, \mathbb{H}$. Arrows in this category correspond exactly to diagrammatic representations of morphisms in a symmetric monoidal category. Since categories of this form exhibit only the identities of various kinds of monoidal categories, they define *free* categories over a graphical signature T.

9.2 Free Monoidal Categories

A monoidal precategory $\mathcal{P}$ consists of a class of objects ob$\mathcal{P}$.

Definition 9.5 (Monoidal Precategory)**.** *Fix a class O and form the free monoid ob$\mathcal{M} := O^*$ of words in O. A* monoidal precategory *is a class of objects ob$\mathcal{M}$ and for every pair $v, w \in O^*$ a set* hom(v, w) *of arrows. A monoidal prefunctor $F : \mathcal{M} \to \mathcal{N}$ consists of a monoid homomorphism ob$\mathcal{M} \to$ ob$\mathcal{N}$ and for every hom-set a function* hom$(v, w) \to$ 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 objects is O and for any pair of words $v, w \in O^*$, the hom-set is formed from the inverse image of T:

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

Definitions 9.6. *Let* $\boldsymbol{TSMC}(T) := \boldsymbol{DCsp}(\boldsymbol{OGraph}_{T_\mathcal{G}}) /\!\!/ \mathbb{H}$. *Let* $\boldsymbol{SMC}(T)$ *be the subcategory of* $\boldsymbol{TSMC}(T)$ *where every graph in the middle of a cospan is directed acyclic.*

$\mathbf{SMC}(T)$ has the property that no graphs contain "feedback loops". Note that T, as a monoidal precategory, embeds canonically into $\mathbf{SMC}(T)$, and hence into $\mathbf{TSMC}(T)$.

Theorem 9.7. $\boldsymbol{SMC}(T)$ *is the free symmetric monoidal category of* T*. That is, for any symmetric monoidal category* $\mathcal{V}$*, any monoidal prefunctor* $F : T \to \mathcal{V}$ *extends uniquely to a symmetric monoidal functor from* $\boldsymbol{SMC}(T)$*. For the embedding of* $T \hookrightarrow \boldsymbol{SMC}(T)$*, there exists a unique monoidal functor* $\hat{F}$ *making the following diagram commute.*

$$\begin{tikzcd}
T \arrow[r, "F"] \arrow[d, hook] & \mathcal{V} \\
\boldsymbol{SMC}(T) \arrow[ur, dashed, "\hat{F}"'] &
\end{tikzcd}$$

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 9.8 (Trace Operator)**.** *For objects* A*,* B *and* C *of* $\boldsymbol{TSMC}(T)$*, a* trace operator *is defined to be a function* $tr^B_{A,C}(-) : \hom_{\boldsymbol{TSMC}(T)}(A \otimes B, C \otimes B) \to \hom_{\boldsymbol{TSMC}(T)}(A, C)$*. Intuitively, this introduces edges that connect from* B *in the codomain to* B *in the domain.*

First, define the graph L_B *as a* $T_\mathcal{G}$*-graph with points* $B + B$ *and exactly one edge connecting each* $b \in B$ *to its copy;* L_B *is simply a collection of edges. Then form a cospan*

$$B \overset{o}{\to} L_B \overset{i}{\leftarrow} B$$

selecting the inputs and outputs of L_B*.*

Let $[\mathcal{G}] : A \otimes B \to C \otimes B$ *be an arrow in* $\boldsymbol{TSMC}(T)$*, represented by a cospan* $\mathcal{G}$*. We can write its arrows as induced arrows of the coproducts, for* $c_1 : C \to G$*,* $c_2 : B \to G$*, etc.*

$$C + B \overset{[c_1,c_2]}{\longrightarrow} G \overset{[d_1,d_2]}{\longleftarrow} A + B$$

Perform the follow (boundary-coherent) pushout of G *and* L_B*, for* $[d_2, c_2]$ *the induced map from the coproduct* $B + B$*.*

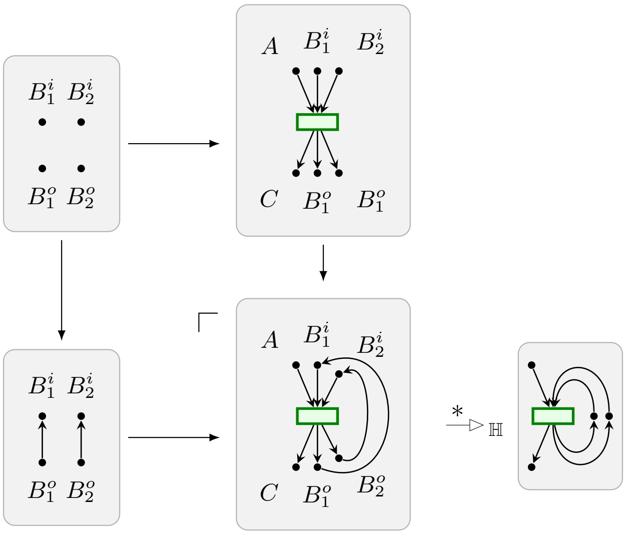
$$\begin{tikzcd}
B + B \arrow[r, "{[d_2, c_2]}"] \arrow[d, "{[o, i]}"'] & G \arrow[d, "p_1"] \\
L_B \arrow[r, "p_2"'] \arrow[ur, phantom, "\ulcorner" very near end] & G'
\end{tikzcd}$$

p_1 *is mono because* $[o, i]$ *is, so let the following be a new cospan* $\mathcal{G}'$*.*

$$C \overset{p_1 c_1}{\longrightarrow} G \overset{p_1 d_1}{\longleftarrow} A$$

Define $tr^B_{A,C}([\mathcal{G}]) := [\mathcal{G}']$*.*

Example 9.9. *An illustration of applying a trace operator.*



This provides a natural way to work 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 9.10. *The trace functional as in 9.8 gives* $\boldsymbol{TSMC}(T)$ *the structure of the free traced symmetric monoidal category over* T.

9.3 PROPs

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

Definition 9.11. *A PROP is a symmetric monoidal category whose objects are the natural numbers 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 9.12. *For a PROP* $\mathbb{P}$ *and some fixed symmetric monoidal category* $\mathcal{V}$*, the category* $\mathbb{P}$*-Alg of* $\mathbb{P}$*-algebras has as objects strict symmetric monoidal functors* $\mathbb{P} \to \mathcal{V}$ *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 $\mathcal{V}$ 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 §9.1. Consider a typed graph category made from a "single-sorted" graphical signature,

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

Then, the objects of $\mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}})$ 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

$$\mathcal{E} := \mathbf{DCsp}(\mathbf{OGraph}_{T_\mathcal{G}}) /\!\!/ (\mathbb{E} + \mathbb{H}).$$

The algebras of $\mathcal{E}$ will be structures in $\mathcal{V}$ 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 $\mathcal{V}$ to be some concrete category like $\mathbf{Vect}_\mathbb{C}$, 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. Methods to support graphical rewriting, using the so called double pushout approach, have also been described, and it has been shown 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 to homeomorphism in topological spaces.

Next, we generalised our construction of open-graphs to work with many *types* of vertices and wires. This makes parameterises open-graphs by a graphical signature with provides the typing rules for how graphs can be composed. This 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. Therefore, we have a fully general method of reasoning about models 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[1], although a proof that this does indeed implement the theory presented here is 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 structure.

Our construction of PROPs and free symmetric monoidal categories is also only the beginning. The graphical notation for traced symmetric monoidal categories introduced in e.g. [Selinger, 2009] gives us strong reason to believe that Conjecture 9.10 is correct. For a suitable notion of traced monoidal theories, generalising the definition of PROPs to the traced setting, we believe that the construction in §9.3 actually forms the *free* traced monoidal theory satisfying the equations reflected by a rewrite system.

On a more fundamental level, the construction of edge-points and edge homeomorphism suggests a deep and telling connection to not only topological graphs, but their more exotic cousins, topological *directed graphs*. This has heretofore 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 by [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 discreet reflection of the deeper, topological theory.

References

[Appelgate et al., 1969] 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*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer Berlin / Heidelberg. 10.1007/BFb0083084.

[Baldan et al., 2008] Baldan, P., Corradini, A., and König, B. (2008). Unfolding graph transformation systems: Theory and applications to verification. In *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, pages 16–36. Springer-Verlag, Berlin, Heidelberg.

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

[Coecke and Duncan, 2008] Coecke, B. and Duncan, R. (2008). Interacting quantum observables. In *ICALP 2008*. LNCS.

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

[1] http://dream.inf.ed.ac.uk/projects/quantomatic.

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

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

[Dixon et al., 2010] Dixon, L., Duncan, R., and Kissinger, A. (2010). Open graphs and computational reasoning. In *Proceedings of DCM'10*, volume 26, pages 169–180. EPTCS.

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

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

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

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

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

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

[Krishnan, 2009] Krishnan, S. (2009). A convenient category of locally preordered spaces. *Applied Categorical Structures*, 17:445–466. 10.1007/s10485-008-9140-9.

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

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

[Lafont, 1990] Lafont, Y. (1990). Interaction nets. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–108, New York, NY, USA. ACM.

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

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

[Lafont and Rannou, 2008] Lafont, Y. and Rannou, P. (2008). Diagram rewriting for orthogonal matrices: A study of critical peaks. In *RTA '08*, pages 232–245, Berlin, Heidelberg. Springer-Verlag.

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

[Penrose, 1971] Penrose, R. (1971). Applications of negative dimensional tensors. In *Combinatorial Mathematics and its Applications*, pages 221–244. Academic Press.

[Prange et al., 2008] 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, 2009] Selinger, P. (2009). A survey of graphical languages for monoidal categories. *New Structures for Physics*, pages 275–337.

A Proof of Freeness for SMC(T)

We shall prove Thm 9.7 using the geometric characterisation of symmetric monoidal categories given in [Joyal and Street, 1991].

Theorem A.1 (9.7)**.** *$\boldsymbol{SMC}(T)$ is the free symmetric monoidal category of T. That is, for any symmetric monoidal category $\mathcal{V}$, any monoidal prefunctor $F : T \to \mathcal{V}$ extends uniquely to a symmetric monoidal functor from $\boldsymbol{SMC}(T)$. For the embedding of $T \hookrightarrow \boldsymbol{SMC}(T)$, there exists a unique monoidal functor $\hat{F}$ making the following diagram commute.*

$$\begin{tikzcd}
T \arrow[r, "F"] \arrow[d, hook] & \mathcal{V} \\
\boldsymbol{SMC}(T) \arrow[ru, dashed, "\hat{F}"'] &
\end{tikzcd}$$

First, we recall several definitions from [Joyal and Street, 1991].

Definition A.2 (Generalised Topological Graph)**.** *A* generalised topological graph *is a pair (G, G_0), where G is a Hausdorff space and G_0 is a discreet, 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.3 (Polarised Graph)**.** *A* polarised graph *is a tuple $\Gamma := (G, G_0, \omega, \pi)$, where ω assigns each each $\hat{e}$ and each circle $\hat{c}$ in (G, G_0) an orientation. 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 $\hat{e}(0) = v$. π then assigns to each v a total order on $in(v)$ and $out(v)$, called a* polarisation. *Also, a polarised graph that contains no directed cycles is called* progressive.

Polarised graphs come with a notion of boundary. We can furthermore put an ordering on this boundary.

Definition A.4 (Boundary of a polarised graph)**.** *For a polarised graph $\Gamma := (G, G_0, \omega, \pi)$, $\partial\Gamma := \hat{G} - G$ is a discreet 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.5 (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 on codomain of arrows in $\mathcal{M}$. A map of anchored graphs with valuations $(\Gamma, v) \to (\Gamma', v')$ 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 $T : A \to O^* \times O^*$ be a graphical signature. $\mathbb{F}_S(T)$ is the category whose objects 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.

It was shown in [Joyal and Street, 1991] that $\mathbb{F}_S(T)$ is the free symmetric monoidal category over T. For the proof of theorem 9.7 it suffices to show that a symmetric monoidal equivalence exists from **SMC**(T) to $\mathbb{F}_S(T)$.

We can now prove Thm 9.7 by defining a geometric realisation functor $[\![-]\!]_T : \mathbf{SMC}(T) \to \mathbb{F}_S(T)$ that is identity-on-objects and showing it admits a (weak) inverse.

Proof. Let $\mathcal{G} : X \to Y$ be an arrow in **SMC**(T). Choose a directed cospan $Y \xrightarrow{c} G \xleftarrow{d} X$ of $T_{\mathcal{G}}$-graphs to represent the equivalence class $\mathcal{G}$.

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

G is an element of the slice category over $T_{\mathcal{G}}$, so it comes with a map $\tau_G : G \to T_{\mathcal{G}}$. The underlying graph of G has an embedding of its boundary and its set of vertices. That is, there exist maps $b : X + Y \to G$ and $v : V \to G$ in **Graph**, where $X + Y$ and V are discreet 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 $T_{\mathcal{G}}$ 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 induce 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 $\tau_G : G \to T_{\mathcal{G}}$, 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 $\mathcal{G}$. 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 $[\![-]\!] : \mathbf{Graph} \to \mathbf{Top}$ 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 $[\![\mathcal{G}]\!]_T = \Gamma$ 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 $\mathcal{G}$ represented by *any* G such that the above construction performed on G yields a progressive anchored graph $\Gamma' \cong \Gamma$. □