# A Framework for Rewriting Families of String Diagrams

Vladimir Zamdzhiev
Université de Lorraine, CNRS, Inria, LORIA, F 54000 Nancy, France

We describe a mathematical framework for equational reasoning about infinite families of string diagrams which is amenable to computer automation. The framework is based on context-free families of string diagrams which we represent using context-free graph grammars. We model equations between infinite families of diagrams using rewrite rules between context-free grammars. Our framework represents equational reasoning about concrete string diagrams and context-free families of string diagrams using double-pushout rewriting on graphs and context-free graph grammars respectively. We prove that our representation is sound by showing that it respects the concrete semantics of string diagrammatic reasoning and we show that our framework is appropriate for software implementation by proving the membership problem is decidable.

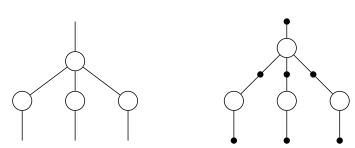
## 1 Introduction

String diagrams have found applications across a range of areas in computer science and related fields such as concurrency [12], systems theory [2], quantum computing [3] and others. A string diagram is a graph-like structure which consists of a collection of *nodes* together with a collection of (possibly open-ended) *wires* connecting nodes to each other (cf. Figure 1a). However, in some application scenarios, it is necessary to reason not just about individual string diagrams, but about entire families of string diagrams (cf. Figure 2). For example, in quantum computing, algorithms and protocols are often described by a family of quantum circuits (diagrams), parameterised by the number of input qubits (wires).

However, as the size of a system grows, reasoning about large (families) of string diagrams becomes cumbersome and error-prone. These issues can be alleviated by using a diagrammatic proof assistant, such as Quantomatic [9], which can automate the reasoning process. Of course, this necessitates developing a formal framework which can represent such families of string diagrams.

We will represent individual string diagrams using *string graphs* [7]. A (directed) string graph is a (directed) graph with two kinds of vertices – *wire* vertices and *node* vertices. Wire vertices have in-degree and out-degree at most one and are used to represent the (open-ended) wires of string diagrams; node vertices can be adjacent only to wire-vertices (cf. Figure 1b).

The primary contribution of this work is to improve the results of [6], which laid the foundation for the representation of *context-free families* of string diagrams and the methods used for equationally reasoning about them. All of the results in this paper are described in detail in the author's PhD thesis [17]. We



(a) (b)

Figure 1: String diagram (a) and its string graph representation (b).

M. Fernández and I. Mackie (Eds.): TERMGRAPH 2018.
EPTCS 288, 2019, pp. 63–76, doi:10.4204/EPTCS.288.6

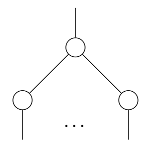


Figure 2: Family of string diagrams.

will represent families of string diagrams and equational reasoning for them by context-free grammars of string graphs and DPO rewriting of these grammars, respectively.

## 2 Background

We begin by introducing some notation and, because of lack of space, briefly recalling the theory of *B-edNCE graph grammars* (see [4] or [17, Chapter 2] for the full definitions). Throughout the rest of the paper, we consider graphs whose vertices are labelled over an alphabet $\Sigma$ and whose edges are labelled over an alphabet $\Gamma$. $\Delta \subseteq \Sigma$ is the *alphabet of terminal vertex labels*; $\mathscr{N} \subseteq \Delta$ is the *alphabet of node-vertex labels* and $\mathscr{W} = \Delta - \mathscr{N}$ is the alphabet of *wire-vertex labels*.

**Definition 1** (Graph [4])**.** A *graph* over an alphabet of vertex labels $\Sigma$ and an alphabet of edge labels $\Gamma$ is a tuple $H = (V, E, \lambda)$, where $V$ is a finite set of nodes, $E \subseteq \{(v, \gamma, w) | v, w \in V, v \neq w, \gamma \in \Gamma\}$ is the set of edges and $\lambda : V \to \Sigma$ is the vertex labelling function.

*Remark* 2. The framework which we describe works with both directed and undirected graphs, where the latter requires only a small simplification of some definitions. To retain generality, all definitions within the paper are stated for directed graphs, but many of our examples show undirected graphs (as they make for more interesting examples). In our notion of graph, self-loops are not allowed and parallel edges are allowed as long as they have distinct labels. This requirement is commonly imposed by node-replacement graph grammars in the litereature.

**Definition 3** (Extended Graph [4])**.** An *Extended Graph* over $\Sigma$ and $\Gamma$ is a pair $(H, C)$, where $H$ is a graph and $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_H \times \{in, out\}$. $C$ is called a *connection relation* and its elements $(\sigma, \beta, \gamma, x, d)$ are called *connection instructions*. The set of all extended graphs over $\Sigma$ and $\Gamma$ is denoted by $EGR_{\Sigma,\Gamma}$.

An *extended graph* provides the necessary information on how a specific graph can be used to replace a nonterminal vertex and connect it to the local neighbourhood of the nonterminal vertex that is to be replaced. In the literature, extended graphs are commonly referred to as *graphs with embedding* [4].

If we are given a mother graph $(H, C_H)$ with a nonterminal vertex $v \in H$ and a daughter graph $(D, C_D)$, then the *substitution* of $(D, C_D)$ for $v$ in $(H, C_H)$, denoted by $(H, C_H)[v/(D, C_D)]$, is given by the extended graph constructed in the following way: for every connection instruction $(\sigma, \beta, \gamma, x, in) \in C_D$ and for every $\sigma$-labelled vertex $w$ in the mother graph for which there is a $\beta$-labelled edge going **in**to the nonterminal vertex $v$ of the mother graph, then the substitution process will establish a $\gamma$-labelled edge from $w$ to $x$. This should become more clear after referring to Example 9. The meaning for $(\sigma, \beta, \gamma, x, out)$ is analogous. Next, we define the concept of an edNCE Graph Grammar. edNCE is an abbreviation for **N**eighbourhood **C**ontrolled **E**mbedding for **d**irected graphs with dynamic **e**dge relabelling.

**Definition 4** (edNCE Graph Grammar [4])**.** An *edNCE Graph Grammar* is given by a pair $G = (P, S)$, where $P$ is a finite set of productions and $S \in \Sigma - \Delta$ is the initial nonterminal label. Productions are of the form $X \to (D, C)$, where $X \in \Sigma - \Delta$ is a nonterminal label and $(D, C) \in EGR_{\Sigma,\Gamma}$ is an extended graph.

For a production $p := X \to (D,C)$, we shall say that the *left-hand side* of $p$ is $X$ and denote it with $lhs(p)$. The *right-hand side* of $p$ is the extended graph $(D,C)$ and we denote it with $rhs(p)$. Vertices which have a label from $\Delta$ are called *terminal vertices* and vertices with labels from $\Sigma - \Delta$ are called *nonterminal vertices*. An (extended) graph is called terminal if all of its vertices are terminal.

Instead of presenting grammars using set-theoretic notation, we will often present them graphically as it is more compact and intuitive. We will use the same notation as in [4], which we now explain. The grammar of Figure 3 has three productions. In each production, the nonterminal symbol is written in the top-left corner of the box. The content of the box is simply the graph which replaces its corresponding nonterminal in a derivation. Any edges crossing the box are the connection instructions which indicate how to connect the graph within the box to its outside context.

**Definition 5** (Extended Graph homomorphism)**.** Given two extended graphs $(H,C_H),(K,C_K) \in EGR_{\Sigma,\Gamma}$, an *extended graph homomorphism* between $(H,C_H)$ and $(K,C_K)$ is a function $f : V_H \to V_K$, such that $f$ is a graph homomorphism from $H$ to $K$ and if $(\sigma,\beta,\gamma,x,d) \in C_H$ then $(\sigma,\beta,\gamma,f(x),d) \in C_K$.

**Definition 6** (Derivation [4])**.** For a graph grammar $G = (P,S)$ and extended graphs $H,H'$, let $v \in V_H$ be a non-terminal vertex with label $X$ and $p : X \to (D,C)$ be a production (copy) of the grammar, such that $H$ and $D$ are disjoint. We say $H \Longrightarrow_{v,p} H'$ is a *derivation step* if $H' = H[v/(D,C)]$. A sequence of derivation steps $H_0 \Longrightarrow_{v_1,p_1} H_1 \Longrightarrow_{v_2,p_2} \cdots \Longrightarrow_{v_n,p_n} H_n$ is called a *derivation*. We write $H \Longrightarrow_* H'$ if there exists a derivation from $H$ to $H'$. A derivation $H \Longrightarrow_* H'$ is *concrete* if $H'$ is terminal.

**Definition 7** (Graph Grammar Language [4])**.** A *sentential form* of an edNCE grammar $G = (P,S)$ is a graph $H$ such that $sn(S,z) \Longrightarrow_* H$ for some $z$, where $sn(S,z)$ is the (extended) graph that has only one vertex given by $z$, its label is $S$ and the graph has no edges and no connection instructions. The graph language induced by $G$ is the set of all terminal sentential forms modulo graph isomorphism.

**Definition 8** (B-edNCE grammar [4])**.** An edNCE grammar $G = (P,S)$ is *boundary*, or a B-edNCE grammar, if for every production $X \to (D,C)$, we have that $D$ contains no edges between nonterminal vertices and $C$ does not contain connection instructions of the form $(\sigma,\beta,\gamma,x,d)$ where $\sigma$ is a nonterminal label.

**Example 9.** The language of the B-edNCE grammar from Figure 3(a) consists of the string graph representations of the string diagrams from Figure 2. The (concrete) derivation which produces the string graph from Figure 1(b) is shown in Figure 3(b).

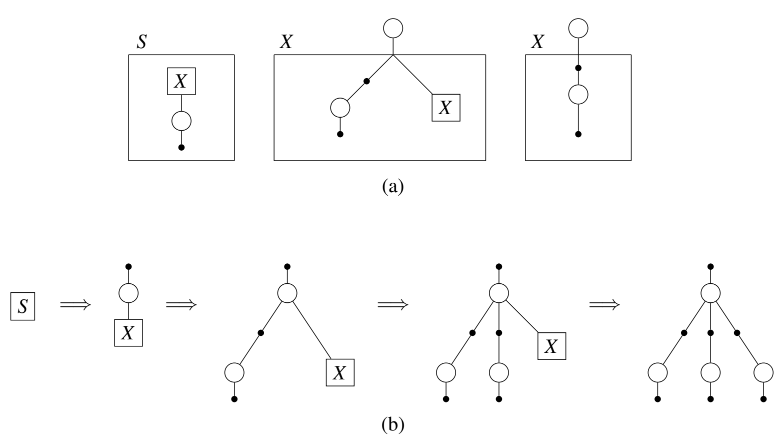



Figure 3: A B-edNCE grammar of undirected graphs (a) and a concrete derivation from it (b).

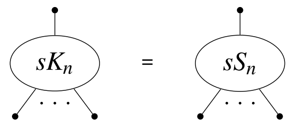



Figure 4: Equational schema

# 3 Reasoning about families of string diagrams

In this section, we present our framework which allows us to equationally reason about entire families of string diagrams (as opposed to individual string diagrams). As a motivating example, we consider the local complementation rule [3] of the ZX-calculus (used in quantum computing), which is crucial for establishing a decision procedure for equality of diagrams in the calculus. The essential data of the rule is given by the equational *schema* in Figure 4. where $sK_n$ is the complete *string* graph on $n$ vertices (consisting of $n$ nodes connected to each other via wires consisting of 1 wire vertex) and $sS_n$ is the star *string* graph on $n$ vertices (the string graph representation of Figure 2). In both cases, every node has also exactly one open-ended wire attached to it.

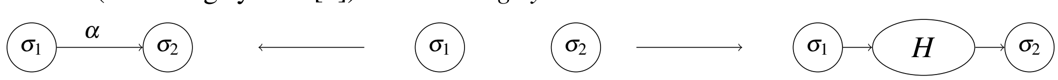
## 3.1 Context-free families of string graphs

As we saw in Example 9, edNCE grammars may represent the $\{sS_n\}_{n\in\mathbb{N}}$ family, but unfortunately, they cannot represent the $\{sK_n\}_{n\in\mathbb{N}}$ family [17, Section 3.3]. However, it is well-known that (B)-edNCE grammars *can* represent the family of complete graphs and if we think of the edges of a complete graph as if they are representing a wire with a single wire vertex, then we can recover the latter family. This idea may be easily formalized and allows us to slightly extend the expressive power of edNCE grammars, while still retaining crucial decidability properties (like the membership problem).

As a first step, we refine the alphabet of edge labels by introducing $\mathscr{E} \subseteq \Gamma$ to be the alphabet of *encoding* edge labels. Essentially, the idea described above is formalized by using graph grammars which generate graphs that contain some encoding edges which are subsequently decoded using a simple confluent and terminating DPO rewrite system.

**Definition 10** (Encoded string graph [6])**.** An *encoded string graph* is a string graph where we additionally allow edges with labels $\alpha \in \mathscr{E}$ to connect pairs of node-vertices. Edges labelled by some $\alpha \in \mathscr{E}$ will be called *encoding* edges.

**Definition 11** (Decoding system [6])**.** A *decoding system* $T$ is a set of DPO rewrite rules of the form:




one for every triple $(\alpha, \sigma_1, \sigma_2) \in \mathscr{E} \times \mathscr{N} \times \mathscr{N}$, where the LHS consists of a single edge with encoding label $\alpha \in \mathscr{E}$ connecting a $\sigma_1$-labelled node-vertex to a $\sigma_2$-labelled node-vertex, and the RHS is a string graph which contains the same two node-vertices and at least one additional vertex while containing no inputs, outputs, or encoding labels.

**Theorem 12.** *Any decoding system $T$ is confluent and terminating.*

Given an (encoded string) graph, *decoding* is the process of applying all of the rules of $T$ to the graph. As the above theorem shows, this is a very simple process which may even be done in a single step. If $H$ is an encoded string graph, we shall say that $H'$ has been *decoded* from $H$, and denote this with $H \Longrightarrow_*^T H'$, if the graph $H'$ is the result of applying all rules from $T$ to $H$, such that $H'$ contains no encoding edges.

**Lemma 13.** *Given two graphs $H, H'$ with $H \Longrightarrow_*^T H'$, where $T$ is a decoding system, then $H$ is an encoded string graph iff $H'$ is a string graph.*

**Definition 14** (Encoded B-edNCE grammar)**.** An *encoded B-edNCE grammar* is a pair $B = (G, T)$, where $G$ is a B-edNCE grammar and $T$ is a decoding system. A *concrete derivation* for an encoded B-edNCE grammar $B = (G, T)$ with $S$ the initial nonterminal, consists of a concrete derivation in $G$ followed by a decoding in $T$, which we denote as $sn(S,z) \Longrightarrow_*^G H_1 \Longrightarrow_*^T H_2$ or simply with $sn(S,z) \Longrightarrow_*^B H_2$ if the graph $H_1$ is not relevant for the context.

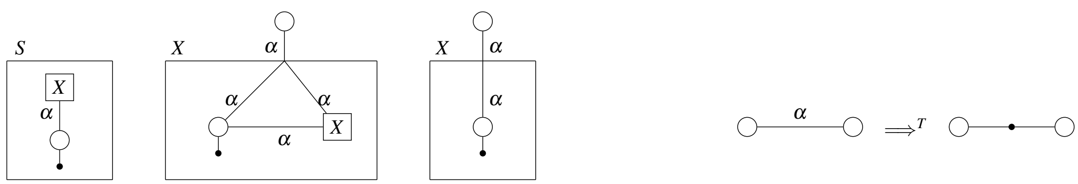
**Definition 15** (B-ESG grammar)**.** A *Boundary Encoded String Graph* (*B-ESG*) grammar is an encoded B-edNCE grammar $B = (G, T)$, subject to some additional coherence conditions, which we omit for lack of space (see [17, Chapter 4]).

The coherence conditions from the above definition are both necessary (up to normal form) and sufficient to generate languages of string graphs, as the next two theorems show.

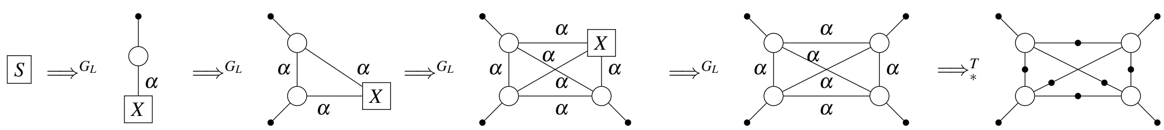
**Theorem 16.** *Every graph in the language of a B-ESG grammar is a string graph.*

**Theorem 17.** *Given an encoded B-edNCE grammar $B = (G, T)$, such that $L(B)$ is a language consisting of string graphs, then there exists a B-ESG grammar $B' = (G', T)$, such that $L(B') = L(B)$. Moreover, $G'$ can be constructed effectively from $G$ and $L(G') = L(G)$.*

**Example 18.** The language of the B-ESG grammar $B = (G, T)$ below is the family $\{sK_n\}_{n \in \mathbb{N}}$.



$sK_4$ is derived by first generating a graph with encoding edges (labelled $\alpha$) followed by decoding:



The grammar from Example 9 is also a B-ESG grammar when equipped with any decoding system (which does not have an effect on the generated language, because the grammar does not contain encoding edges).

**Theorem 19.** The membership problem for B-ESG grammars is decidable.

**Definition 20** (Grammar homomorphism)**.** Given two edNCE grammars $G_1 = (P_1, S_1)$ and $G_2 = (P_2, S_2)$, a *grammar homomorphism* from $G_1$ to $G_2$ is a function $m : P_1 \to P_2$, together with a collection of extended graph homomorphisms (cf. Definition 5) $m_{p_i} : rhs(p_i) \to rhs(m(p_i))$ one for each production $p_i \in P_1$, such that $lhs(p_i) = lhs(m(p_i))$.

**Definition 21** (Category of B-ESG grammars)**.** The category of B-ESG grammars over a decoding system $T$, denoted $\mathbf{B\text{-}ESG}_T$, or simply **B-ESG** if $T$ is clear from the context, has objects B-ESG grammars $B = (G, T)$. A morphism $h$ between two B-ESG grammars $B_1 = (G_1, T)$ and $B_2 = (G_2, T)$ is a grammar homomorphism $h : G_1 \to G_2$.

**Theorem 22.** *$\mathbf{B\text{-}ESG}_T$ is a* partially adhesive *category (cf. [7]).*

The above theorem means that DPO rewriting of B-ESG grammars *themselves* is well-behaved, provided that some additional matching conditions (which are fully characterised) are satisfied [17, Chapter 5].

### 3.2 Rewrite schemas for families of string graphs

We saw how to represent families of string graphs, next we explain how to represent equational schemas between such families. That is, we wish to establish a constructive bijection between the graphs of one family and the graphs of the other. We may do so if we require that the pair of B-ESG grammars have a 1-1 correspondence between their productions and so do the nonterminal vertices in corresponding productions. This allows us to perform *parallel* concrete derivations between the two grammars and we may thus establish the constructive bijection we required.

**Definition 23** (Extended graph rewrite rule)**.** An *extended graph rewrite rule* is a pair of monomorphisms $L \xleftarrow{l} I \xrightarrow{r} R$, where all objects are extended graphs.

**Definition 24.** Given extended graphs $G$ and $D$ where $x \in V_G$ is a (nonterminal) vertex, and given monomorphisms $m_1 : G \to G'$ and $m_2 : D \to D'$, then

$$m_3(v) := \begin{cases} m_1(v) & \text{if } v \in V_G \\ m_2(v) & \text{if } v \in V_D \end{cases}$$

is the induced *substituted monomorphism* $m_3 : G[x/D] \to G'[m_1(x)/D']$, which we denote by $SM(m_1, m_2, x)$.

**Definition 25** (Rewrite rule substitution)**.** Given extended graph rewrite rules $B_1 := L_1 \xleftarrow{l_1} I_1 \xrightarrow{r_1} R_1$ and $B_2 := L_2 \xleftarrow{l_2} I_2 \xrightarrow{r_2} R_2$, with (non-terminal) vertex $v \in I_1$ then the *substitution* of $B_2$ for $v$ in $B_1$, denoted $B_1[v/B_2]$ is given by the extended graph rewrite rule $B_3 := L_3 \xleftarrow{l_3} I_3 \xrightarrow{r_3} R_3$, where $L_3 := L_1[l_1(v)/L_2]$, $I_3 := I_1[v/I_2]$, $R_3 := R_1[r_1(v)/R_2]$, $l_3 := SM(l_1, l_2, v)$ and $r_3 := SM(r_1, r_2, v)$.

**Definition 26** (B-edNCE Pattern)**.** A B-edNCE *pattern* is a triple of B-edNCE grammars $B := G_L \xleftarrow{l} G_I \xrightarrow{r} G_R$, where $l$ and $r$ are grammar monomorphisms which are bijections between the productions of all three grammars. Moreover, $l$ and $r$ are also label-preserving bijections between the nonterminals in corresponding productions of the grammars. In addition, all three grammars have the same initial nonterminal label. If $p$ is a production in $G_L$, then $B_p$ will refer to the extended graph rewrite rule $rhs(p) \xleftarrow{l_{p'}} rhs(p') \xrightarrow{r_{p'}} rhs(p'')$, where $p'$ and $p''$ are the corresponding productions of $p$ in $G_I$ and $G_R$ respectively, and $l_{p'}$ and $r_{p'}$ are the components (cf. Definition 20) of the monomorphisms $l$ and $r$ at production $p'$.

These conditions ensure that we may perform parallel derivations in the sense that at each step we apply corresponding productions and replace corresponding nonterminals in all three grammars.

**Definition 27** (B-edNCE Pattern Instantiation)**.** Given a B-edNCE pattern $G_L \xleftarrow{l} G_I \xrightarrow{r} G_R$, a parallel instantiation is a triple of concrete derivation sequences of the following form:

$$\begin{aligned}
&sn(S, v_1) \Longrightarrow^{G_L}_{v_1, l(p_1)} H'_1 \Longrightarrow^{G_L}_{l(v_2), l(p_2)} H'_2 \Longrightarrow^{G_L}_{l(v_3), l(p_3)} \cdots \Longrightarrow^{G_L}_{l(v_n), l(p_n)} H'_n \\
&sn(S, v_1) \Longrightarrow^{G_I}_{v_1, p_1} H_1 \Longrightarrow^{G_I}_{v_2, p_2} H_2 \Longrightarrow^{G_I}_{v_3, p_3} \cdots \Longrightarrow^{G_I}_{v_n, p_n} H_n \\
&sn(S, v_1) \Longrightarrow^{G_R}_{v_1, r(p_1)} H''_1 \Longrightarrow^{G_R}_{r(v_2), r(p_2)} H''_2 \Longrightarrow^{G_R}_{r(v_3), r(p_3)} \cdots \Longrightarrow^{G_R}_{r(v_n), r(p_n)} H''_n
\end{aligned}$$

The language of $B$, denoted $L(B)$, is the set of all graph rewrite rules $H'_n \xleftarrow{l_n} H_n \xrightarrow{r_n} H''_n$ obtained by performing concrete parallel derivations, where $l_n$ and $r_n$ are induced by the derivation process.

**Definition 28** (Production input/output/isolated vertex)**.** Given a B-ESG grammar $B$, we say that a wire-vertex $w$ is a *production input (output)* if its in-degree (out-degree) is zero and it has no incoming (outgoing) connection instructions. $w$ is a *production isolated wire-vertex* if it is both a production input and a production output.

**Definition 29** (B-ESG rewrite rule)**.** A *B-ESG rewrite rule* is a span of monos $B_L \xleftarrow{l} B_I \xrightarrow{r} B_R$, where $B_L = (G_L, T), B_I = (G_I, T), B_R = (G_R, T)$, such that $G_L \xleftarrow{l} G_I \xrightarrow{r} G_R$ is a B-edNCE pattern such that for every triple of corresponding productions $p_L, p_I, p_R$ in $G_L, G_I, G_R$ respectively, we have:

**Boundary:** $p_I$ contains only nonterminal vertices and isolated wire-vertices and it contains no edges, connection instructions or node-vertices.

**IO1:** $l$ and $r$ are surjections on the production inputs (outputs) between $p_I$ and $p_L$, $p_I$ and $p_R$ respectively.

**IO2:** For every wire-vertex $w \in p_I$, $l(w)$ and $r(w)$ are both a production input (output) in $p_L$ and $p_R$ respectively.

Moreover, the grammars $G_I, G_L, G_R$ must be in a certain normal form (cf. [17, Section 5.3]).

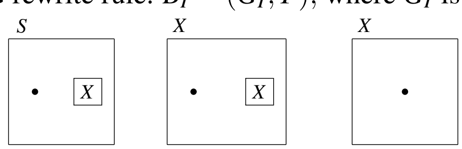
**Definition 30** (B-ESG Rewrite Rule Instantiation)**.** Given a B-ESG rewrite rule $B := B_L \xleftarrow{l} B_I \xrightarrow{r} B_R$, a parallel instantiation is a B-edNCE pattern instantiation for $G_L \xleftarrow{l} G_I \xrightarrow{r} G_R$ followed by a decoding:

$$sn(S, v_1) \Longrightarrow^{G_L}_{v_1, l(p_1)} H_1' \Longrightarrow^{G_L}_{l(v_2), l(p_2)} H_2' \Longrightarrow^{G_L}_{l(v_3), l(p_3)} \cdots \Longrightarrow^{G_L}_{l(v_n), l(p_n)} H_n' \Longrightarrow^{T}_{*} F'$$
$$sn(S, v_1) \Longrightarrow^{G_I}_{v_1, p_1} H_1 \Longrightarrow^{G_I}_{v_2, p_2} H_2 \Longrightarrow^{G_I}_{v_3, p_3} \cdots \Longrightarrow^{G_I}_{v_n, p_n} H_n \Longrightarrow^{T}_{*} F$$
$$sn(S, v_1) \Longrightarrow^{G_R}_{v_1, r(p_1)} H_1'' \Longrightarrow^{G_R}_{r(v_2), r(p_2)} H_2'' \Longrightarrow^{G_R}_{r(v_3), r(p_3)} \cdots \Longrightarrow^{G_R}_{r(v_n), r(p_n)} H_n'' \Longrightarrow^{T}_{*} F''$$

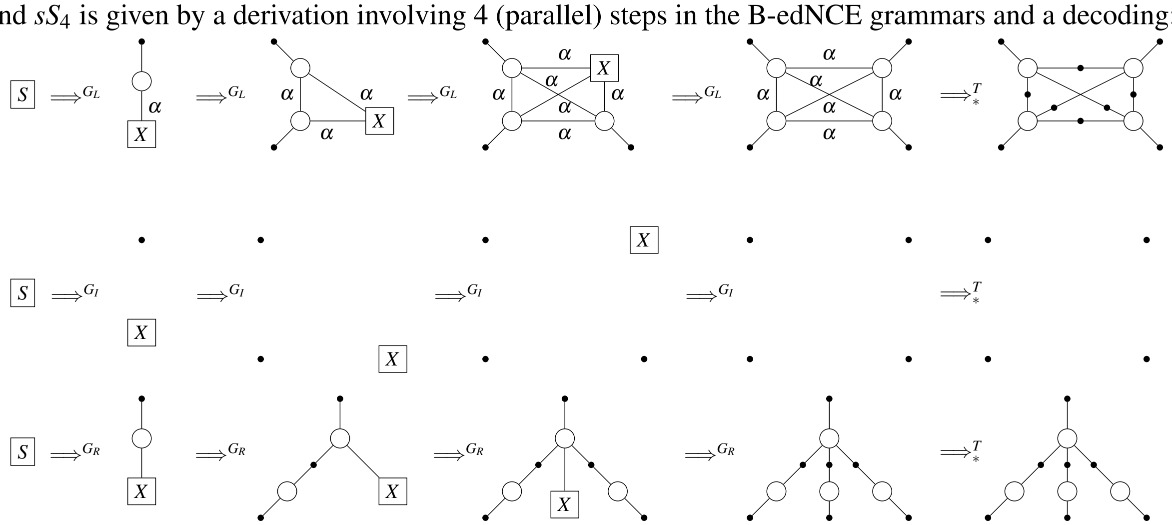
The language of $B$, denoted $L(B)$, is the set of all rewrite rules $F' \xleftarrow{l_F} F \xrightarrow{r_F} F''$ obtained by performing parallel derivations, where the embeddings $F' \xleftarrow{l_F} F \xrightarrow{r_F} F''$ are induced by the derivation process.

**Theorem 31.** *The language of every B-ESG rewrite rule consists solely of string graph rewrite rules.*

**Example 32.** Let $B_L = (G_L, T)$ be the B-ESG grammar representing $\{sK_n\}_{n \in \mathbb{N}}$ from Example 18 and let $B_R = (G_R, T)$, where $G_R$ is the grammar representing $\{sS_n\}_{n \in \mathbb{N}}$ from Figure3. Given this data, there is a unique (and constructive) choice for a B-ESG grammar $B_I$ and embeddings $l, r$, such that $B := B_L \xleftarrow{l} B_I \xrightarrow{r} B_R$ is a B-ESG rewrite rule. $B_I = (G_I, T)$, where $G_I$ is given by:




and $l$ and $r$ are the obvious grammar embeddings. A derivation of a string graph rewrite rule relating $sK_4$ and $sS_4$ is given by a derivation involving 4 (parallel) steps in the B-edNCE grammars and a decoding:



where there are obvious induced embeddings of the middle string graph into the other two (details omitted).

### 3.3 B-ESG grammar rewriting

Next, we show how to model equational reasoning by applying an equational schema to a context-free family of diagrams. We begin by first showing that DPO rewriting (which we use to model equational reasoning for graphs and grammars) behaves well with respect to graph substitution (which we use for language generation). We use the notation $H \rightsquigarrow_B^m H'$ to indicate that the DPO rewrite of the (extended) graph $H$ using the (extended) graph rewrite rule $B$ at matching $m$ is the (extended) graph $H'$.

**Theorem 33.** *Given boundary extended graphs $H, H', D, D'$, such that $H \rightsquigarrow_{B_1}^{m_1} H'$ and $D \rightsquigarrow_{B_2}^{m_2} D'$, where $B_1 := L_1 \xleftarrow{l_1} I_1 \xrightarrow{r_1} R_1$, $B_2 := L_2 \xleftarrow{l_2} I_2 \xrightarrow{r_2} R_2$, $v \in V_{I_1}$ and where $m_1, m_2$ are matchings (subject to some additional conditions), then $H[m_1 \circ l_1(v)/D] \rightsquigarrow_{B_3}^{m_3} H'[f_1 \circ r_1(v)/D']$, where $B_3 := B_1[v/B_2]$ and $m_3 = SM(m_1, m_2, l_1(v))$. In terms of diagrams, given the following two DPO rewrites:*

$$
\begin{array}{ccccc}
L_1 & \xleftarrow{l_1} & I_1 & \xrightarrow{r_1} & R_1 \\
{\scriptstyle m_1}\downarrow & & {\scriptstyle k_1}\downarrow & & \downarrow{\scriptstyle f_1} \\
H & \xleftarrow[s_1]{} & K_1 & \xrightarrow[g_1]{} & H'
\end{array}
\qquad
\begin{array}{ccccc}
L_2 & \xleftarrow{l_2} & I_2 & \xrightarrow{r_2} & R_2 \\
{\scriptstyle m_2}\downarrow & & {\scriptstyle k_2}\downarrow & & \downarrow{\scriptstyle f_2} \\
D & \xleftarrow[s_2]{} & K_2 & \xrightarrow[g_2]{} & D'
\end{array}
$$

*then the following diagram is also a DPO rewrite:*

$$
\begin{array}{ccccc}
L_1[l_1(v)/L_2] & \xleftarrow{l_3} & I_1[v/I_2] & \xrightarrow{r_3} & R_1[r_1(v)/R_2] \\
{\scriptstyle m_3}\downarrow & & {\scriptstyle k_3}\downarrow & & \downarrow{\scriptstyle f_3} \\
H[m_1 \circ l_1(v)/D] & \xleftarrow[s_3]{} & K_1[k_1(v)/K_2] & \xrightarrow[g_3]{} & H'[f_1 \circ r_1(v)/D']
\end{array}
$$

*where each $x_3$ is the obvious substituted monomorphism (cf. Definition 24).*

**Definition 34** (B-ESG rewrite)**.** Given a B-ESG rewrite rule $B = B_L \xleftarrow{l} B_I \xrightarrow{r} B_R$ with initial nonterminal label $S$ and a B-ESG grammar $B_H$, such that $B_H$ is in normal form (cf. [17, Section 5.3]), then we will say that the *B-ESG rewrite* of $B_H$ using $B$ over a matching $m : G_L \to G_H$, is the B-ESG grammar $B_M = (G_M, T)$, denoted by $B_H \rightsquigarrow_{B,m} B_M$, where $G_M$ is given by the DPO rewrite:

$$
\begin{array}{ccccc}
G_L & \xleftarrow{l} & G_I & \xrightarrow{r} & G_R \\
{\scriptstyle m}\downarrow & & {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle f} \\
G_H & \xleftarrow[s]{} & G_K & \xrightarrow[g]{} & G_M
\end{array}
$$

**Theorem 35.** *Given a B-ESG rewrite $B_H \rightsquigarrow_{B,m} B_M$, as in Definition 34, where the matching $m$ satisfies some additional conditions (cf. [17, Section 5.4]), then the rewrite $B_H \rightsquigarrow_{B,m} B_M$ is admissible with respect to $L(B)$ in the following sense: if $(K, K')$ is a parallel instantiation (cf. Definition 30) of $(B_H, B_M)$, then there exists a sequence of string graph rewrite rules $s_1, \ldots, s_n \in L(B)$, such that $K \rightsquigarrow_{s_1} \cdots \rightsquigarrow_{s_n} K'$.*

**Example 36.** Consider the equational schema in Figure 5. It can be derived from its left-hand side by applying the equational schema from Figure 4. Moreover, this application respects the concrete semantics of the graph families in the sense that every *instantiation* of the former schema can be obtained

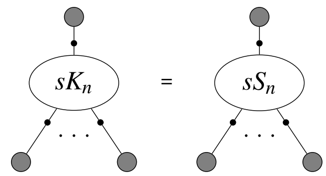
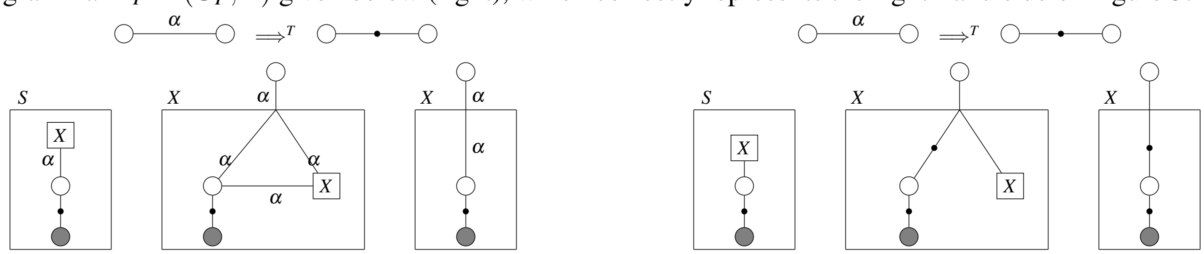



Figure 5: Derived equational schema.

by applying a specific *instantiation* of the latter schema. We now show how to model this using our framework. The left-hand side of Figure 5 is represented by the B-ESG grammar $B_H = (G_H, T)$ given below (left). By performing a B-ESG rewrite using the B-ESG rewrite rule from Example 32, we get the grammar $B_P = (G_P, T)$ given below (right), which correctly represents the right-hand side of Figure 5.

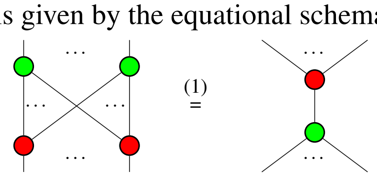


Moreover, this rewrite respects the concrete semantics, because for any parallel instantiation $(K, K')$ of the two grammars above, the string graph $K$ may be rewritten into the string graph $K'$ by applying a concrete string graph rewrite rule induced by the B-ESG rewrite rule from Example 32.

## 4 Use case: generalised bialgebra rule

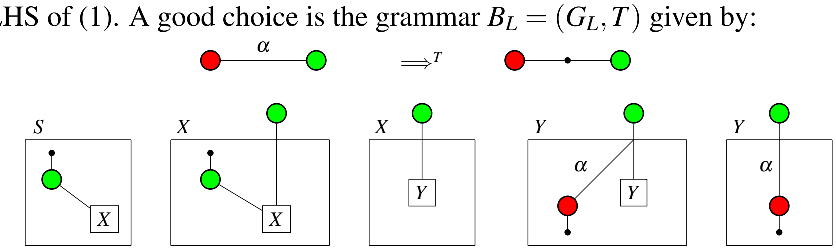
In the previous section we presented a running example which showed how to model the local complementation rule for the ZX-calculus. We now provide another example, namely of the Generalised Bialgebra rule, which is an important distributivity law for several diagrammatic calculi, including the ZX-calculus. We will show how this can be represented using our framework.

The generalised bialgebra rule is given by the equational schema of string diagrams given by (1):

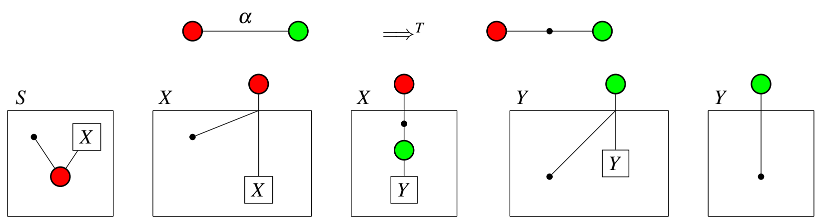



where the LHS has $m$ green nodes each of which is connected to all $n$ red nodes and, in addition, all green and red nodes have exactly one input/output wire. The inputs are the open-ended wires at the bottom and the outputs are the open-ended wires at the top of the diagram. We will call this family of string diagrams $sK_{m,n}$. The RHS family we call $sS_{m,n}$ and it (necessarily) has $m$ inputs and $n$ outputs.

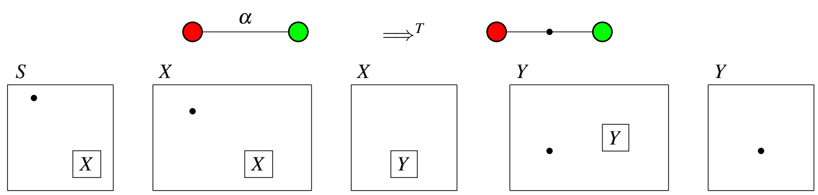
In order to represent this rule using our framework, we first have to describe a B-ESG grammar that describes the LHS of (1). A good choice is the grammar $B_L = (G_L, T)$ given by:



Next, we have to represent the RHS of (1). We remark that our choice of grammar has to be consistent with the one just constructed in order to satisfy the requirements of our framework. A suitable choice is the grammar $B_R = (G_R, T)$ given by:

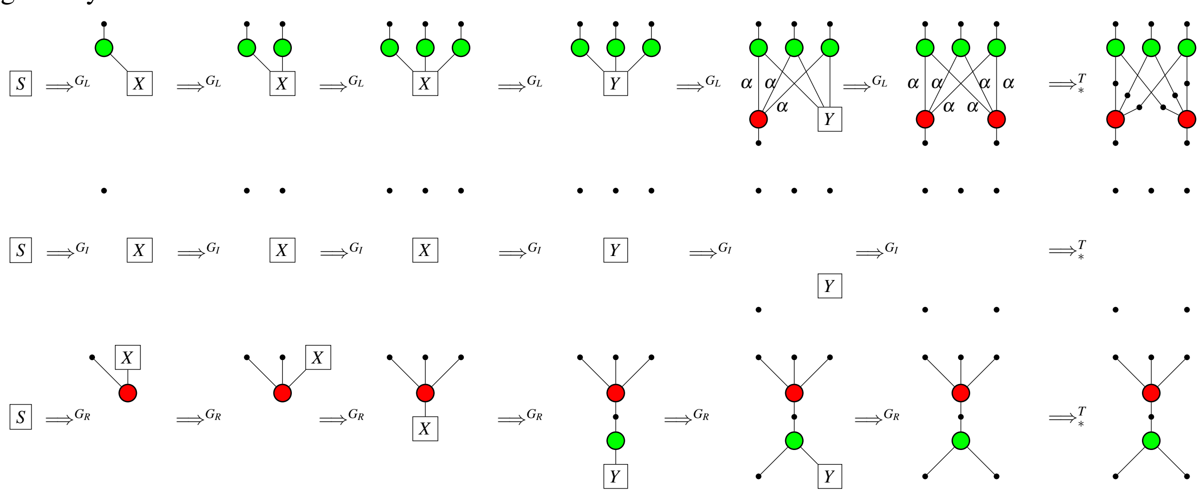



Our framework ensures that we may now uniquely construct an interface grammar by just taking the nonterminal vertices and inputs/outputs from each production of either grammar $B_L$ or $B_R$. We name the resulting grammar $B_I = (G_I, T)$ :

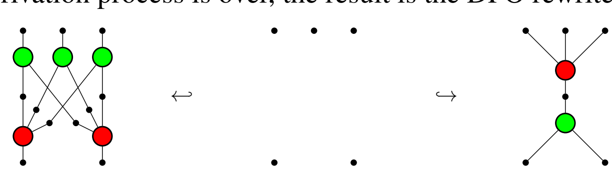



There are now obvious grammar embeddings $l$ and $r$, such that $B := (B_L \xleftarrow{l} B_I \xrightarrow{r} B_R)$ is a B-ESG rewrite pattern. The pattern $B$ therefore encodes the rewrite rule $sK_{m,n} \rightsquigarrow sS_{m,n}$. Of course, by swapping $B_L$ and $B_R$ we may represent the other direction of equation (1), but we shall only consider the left-to-right direction in this example.

Our framework guarantees that for any specific choice of $m$ and $n$, we get a string graph DPO rewrite rule $sK_{m.n} \rightsquigarrow sS_{m,n}$. The parallel derivation of $B$ which produces the DPO rewrite rule $sK_{3,2} \rightsquigarrow sS_{3,2}$ is given by:

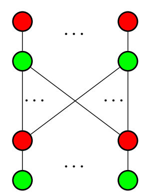


Observe that the middle derivation is uninteresting, because it is uniquely determined by either of the other two derivations by just taking the nonterminal vertices and the inputs/outputs in corresponding productions. Once the derivation process is over, the result is the DPO rewrite rule $sK_{3,2} \rightsquigarrow sS_{3,2}$:

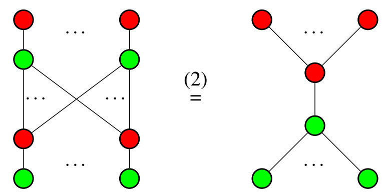


where the obvious embeddings are induced by the derivation process of the grammar. Again, observe that the interface of the DPO rewrite rule is unqiuely determined by either the left or right graph by taking the inputs/outputs.

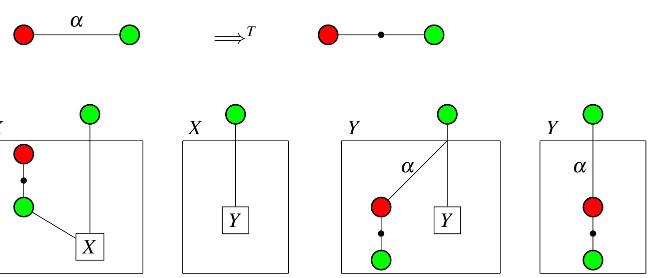
Next, we illustrate how our framework supports rewriting of families of string diagrams. Consider the family of string diagrams shown below:



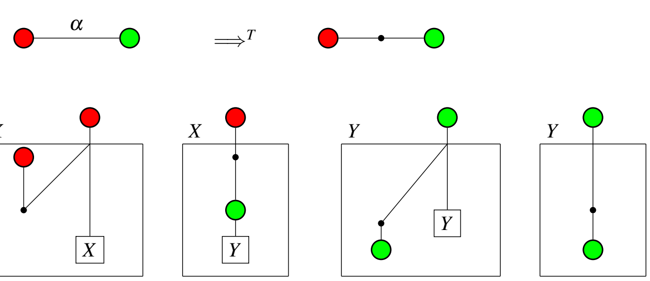
If we apply the equational schema (1) to it, then we get the derived equational schema (2):



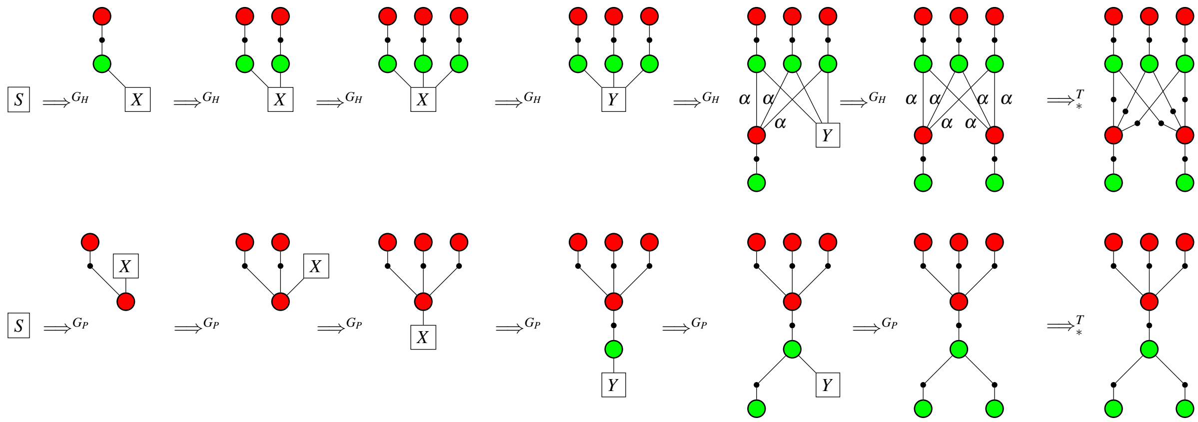

To represent this rewrite in our framework, we start by constructing a B-ESG grammar for the LHS of (2). We choose the grammar $B_H = (G_H, T)$ as follows:




The previously constructed *grammar* rewrite rule $B$ may now be matched into $B_H$ and the DPO rewrite can be performed, yielding the grammar $B_P = (G_P, T)$ :




Observe that $B_P$ correctly represents the RHS of (2) and that the *derived* rewrite pattern $B' := (B_H \hookleftarrow B_I \hookrightarrow B_P)$ correctly represents the *derived* equational schema (2). More concretely, because the match satisfies the strong requirements imposed by our framework, the performed rewrite is admissible with respect to $B$ in the following sense. For any parallel derivation $(B_H, B_P) \Longrightarrow_* (H, P)$, the string graph $H$ can be rewritten to the string graph $P$ using a sequence of DPO rewrite rules induced by $B$ (in this specific case, this may be done using a single DPO rule). For instance, consider the parallel derivation of $(B_H, B_P) \Longrightarrow_* (H_{3,2}, P_{3,2})$:




Then string graph rewrite rule $sK_{3,2} \rightsquigarrow sS_{3,2}$ from above (which is induced by $B$) can be matched into the string graph $H_{3,2}$ and the DPO rewrite yields the string graph $P_{3,2}$. In this way, our framework soundly models equational reasoning with families of string diagrams.

**Related work.** B-ESG rewrite patterns are similar to the *pair grammars* approach presented in [13]. In that paper the author defines a pair of graph grammars whose productions are in bijection which moreover preserves the nonterminals within them. As a result, parallel derivations are defined in a similar way to our B-ESG rewrite patterns. However, the author uses a different notion of grammar which is less expressive than ours.

The pair grammars approach has inspired the development of *triple graph grammars* [16]. In this approach, the author uses a triple of grammars $(L,C,R)$, which also share a bijective correspondence between their productions. In this sense, they are similar to our B-ESG rewrite rules. However, the middle grammar $C$ is used to relate graph elements from $L$ to graph elements of $R$ in a more powerful way compared to our approach. We simply use the middle grammar in order to identify the interface and interior elements for performing DPO rewrites. However, the grammar model used in [16] is based on monotonic single-pushout (SPO) productions with no notion of nonterminal elements. These grammars are not expressive enough for our purposes.

Another way of formalising families of string diagrams is by using *!-graphs* [5, 11]. !-graphs have a considerably simpler graphical presentation compared to B-ESG grammars. This is the underlying mechanism which Quantomatic uses for representing families of string diagrams. Unfortunately, !-graphs also have somewhat limited expressive power. In fact, the original motivation for developing B-ESG grammars was to address these shortcomings. Detailed comparisons in terms of expressivity of the two formalisms are available in [17, 8].

A third way of representing families of string diagrams is by using a programming language designed to generate such diagrams. Examples include Proto-Quipper-M [15, 10] and EWire [14]. Even though these languages have not been studied in terms of their formal expressive power, it seems very likely that they both have higher expressive power, but worse decidability properties compared to !-graphs and B-ESG grammars (e.g. the membership problem is unlikely to be decidable).

## 5 Conclusion and future work

We introduced B-ESG grammars which are slightly extended context-free graph grammars that generate string graphs and which therefore represent families of string diagrams. We showed that by carefully relating the productions of a triple of grammars we are able to correctly represent equational schemas of string diagrams. The category of B-ESG grammars enjoys a partial adhesive structure and DPO rewriting in that category is not only well-behaved, but also admissible with respect to the derivation process of the grammars, provided that strong matching conditions are satisfied, which shows that our framework soundly models equational reasoning about families of string diagrams.

Our framework represents string diagrams as string *graphs*. More recently, a new representation of string diagrams using *hypergraphs* has been proposed [1] which has a simpler and more elegant meta-theory (e.g. no need to quotient wire-vertices on wires) and which also enjoys better categorical properties (e.g. adhesivity vs partial adhesivity). As part of future work, we will consider developing a new framework which can represent families of string diagrams using *hypergraph grammars*, which also enjoy better structural and algebraic properties compared to B-edNCE grammars. The crucial ideas presented in this paper should carry through straightforwardly to the hypergraph setting, but it remains to be seen whether the hypergraph representation has adequate expressive power in terms of the languages it can generate.

**Acknowledgements.** I would like to thank Aleks Kissinger for many fruitful discussions and also for his supervision of my PhD studies, during which time these results were discovered. I also thank my other former supervisors, Samson Abramsky and Bob Coecke, for providing excellent advice during this time.

## References

[1] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński & F. Zanasi (2016): *Rewriting Modulo Symmetric Monoidal Structure*. LICS '16, doi:10.1145/2933575.2935316.

[2] F. Bonchi, P. Sobocinski & F. Zanasi (2015): *Full Abstraction for Signal Flow Graphs*. In: *POPL 2015*, doi:10.1145/2676726.2676993.

[3] Ross Duncan & Simon Perdrix (2009): *Graph States and the Necessity of Euler Decomposition*. In: *Mathematical Theory and Computational Practice*, doi:10.1007/978-3-642-03073-4_18.

[4] J. Engelfriet & G. Rozenberg (1997): *Node Replacement Graph Grammars*. In Grzegorz Rozenberg, editor: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, doi:10.1142/9789812384720_0001.

[5] A. Kissinger, A. Merry & M. Soloviev (2012): *Pattern graph rewrite systems*. In: 8th International Workshop on *Developments in Computational Models*, doi:10.4204/EPTCS.143.5.

[6] A. Kissinger & V. Zamdzhiev (2015): *Equational Reasoning with Context-Free Families of String Diagrams*. In: *Graph Transformation - 8th International Conference, ICGT 2015*, doi:10.1007/978-3-319-21145-9_9.

[7] Aleks Kissinger (2012): *Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing*. PhD thesis. Oxford University. `http://arxiv.org/abs/1203.0202`.

[8] Aleks Kissinger & Vladimir Zamdzhiev (2015): *!-graphs with trivial overlap are context-free*. In: *Proceedings Graphs as Models, GaM 2015, London, UK, 11-12 April 2015.*, doi:10.4204/EPTCS.181.2.

[9] Aleks Kissinger & Vladimir Zamdzhiev (2015): *Quantomatic: A Proof Assistant for Diagrammatic Reasoning*. In: *CADE 2015*, doi:10.1007/978-3-319-21401-6_22.

[10] Bert Lindenhovius, Michael Mislove & Vladimir Zamdzhiev (2018): *Enriching a Linear/Non-linear Lambda Calculus: A Programming Language for String Diagrams*. In: *LICS 2018*, doi:10.1145/3209108.3209196.

[11] A. Merry (2014): *Reasoning with !-Graphs*. DPhil Thesis, University of Oxford, `http://arxiv.org/abs/1403.7828`.

[12] J. Meseguer & U. Montanari (1988): *Petri Nets Are Monoids: A New Algebraic Foundation for Net Theory*. In: *LICS 1988*, doi:10.1109/LICS.1988.5114.

[13] Terrence W. Pratt (1971): *Pair grammars, graph languages and string-to-graph translations*. Journal of Computer and System Sciences 5(6), pp. 560 – 595, doi:10.1016/S0022-0000(71)80016-8.

[14] Mathys Rennela & Sam Staton (2018): *Classical Control and Quantum Circuits in Enriched Category Theory*. Electr. Notes Theor. Comput. Sci. 336, pp. 257–279, doi:10.1016/j.entcs.2018.03.027.

[15] Francisco Rios & Peter Selinger (2017): *A categorical model for a quantum circuit description language*. In: QPL 2017., doi:10.4204/EPTCS.266.11.

[16] Andy Schürr (1995): *Specification of graph translators with triple graph grammars*. In: Graph-Theoretic Concepts in Computer Science, doi:10.1007/3-540-59071-4_45.

[17] V. Zamdzhiev (2016): *Rewriting context-free families of string diagrams*. Ph.D. thesis, University of Oxford. Available at `https://ora.ox.ac.uk/objects/uuid:4138e2fe-5382-429b-a3f6-67f8770e2cd3`.