REWRITING STRUCTURED COSPANS

DANIEL CICALA

Abstract. To foster the study of networks on an abstract level, we further study the formalism of *structured cospans* introduced by Baez and Courser. A structured cospan is a diagram of the form $La \to x \leftarrow Lb$ built from a geometric morphism with left exact left adjoint $L \dashv R \colon \mathsf{X} \to \mathsf{A}$. We show that this construction is functorial and results in a topos with structured cospans for objects. Additionally, structured cospans themselves are compositional. Combining these two perspectives, we define a double category of structured cospans. We then leverage adhesive categories to create a theory of rewriting for structured cospans. We generalize the result from graph rewriting stating that a graph grammar induces the same rewrite relation as its underlying graph grammar. We use this fact to prove our main result, a complete characterization of the rewriting relation for a topos X using double categories. This provides a compositional framework for rewriting systems.

1. Introduction

2. Rewriting in topoi

(Outline of this section
(a) retell the story of DPO rewriting with the scope restricted to topoi. Say it's included merely for completeness. Point to LackSobo and others for detailed story. Encompass rewrite rules, grammars, language.
(b) Discuss language hierarchy and expressiveness a la Chomsky and Ehrig, et al. Show that discrete grammar is as expressive as the non-discrete.
)

2.1. **Quickly sketching the basics.** Fix a topos T. Rewriting starts with the notion of a **rewrite rule**, or simply **rule**. In its most general form, a rule is a span

$$\ell \leftarrow k \rightarrow r$$

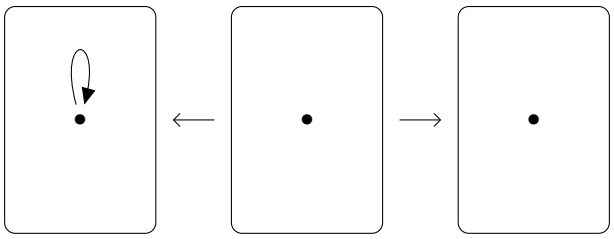
in T. The arrows are left unnamed unless we need to refer to them. For us, rules come in two flavors. A **fine rule** is one in which both of the span arrows are monic. A **bold rule** is one without restriction on the arrows.

Remark 2.1. Both fine and bold approaches are considered in the rewriting literature, but often by the name 'linear' and 'non-linear', respectively. Fine rewriting is more common. Habel, Muller, and Plump compared these alternatives in the context of graph rewriting [**?**]. The distinction between the two cases does not appear in this chapter, and everything we say carries through in either case. We do take care to ensure that constructions are well-defined in the monic case.

Date: September 21, 2019.

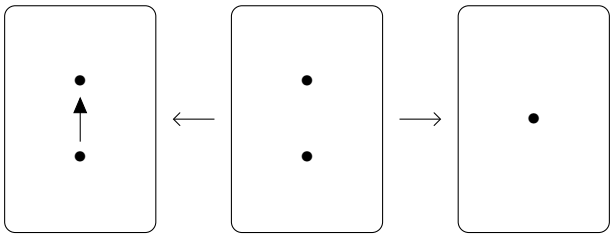
The conceit of a rule is that r replaces ℓ while k identifies a subsystem of ℓ that remains fixed. For example, suppose we were modeling some system using graphs where self-loops were meaningless. In the introduction, we considered modeling the internet with a graph with websites as nodes and links as edges. If we did not care about websites with a link to itself, we would introduce a rule that replaces a node with a loop with a node

(1)



For another example, suppose we had another system modeled on graphs where an edge between two nodes is equivalent to having a single node. This is captured with the rule

(2)



This rule appears in the ZX-calculus example from Section **??**. Observe that the first example is a fine rewrite and the second is a bold rewrite.

To *apply* a rule $\ell \leftarrow k \rightarrow r$ to an object g, we require an arrow $m\colon \ell \rightarrow g$ such that there exists a **pushout complement**, an object d fitting into a pushout diagram

$$\begin{array}{ccc} \ell & \longleftarrow & k \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow \\ g & \longleftarrow & d \end{array}$$

A pushout complement need not exist, but when it does and the map $k \rightarrow \ell$ is monic, then it is unique up to isomorphism [**?**, Lem. 15].

For each application of a rule, we derive a new rule.

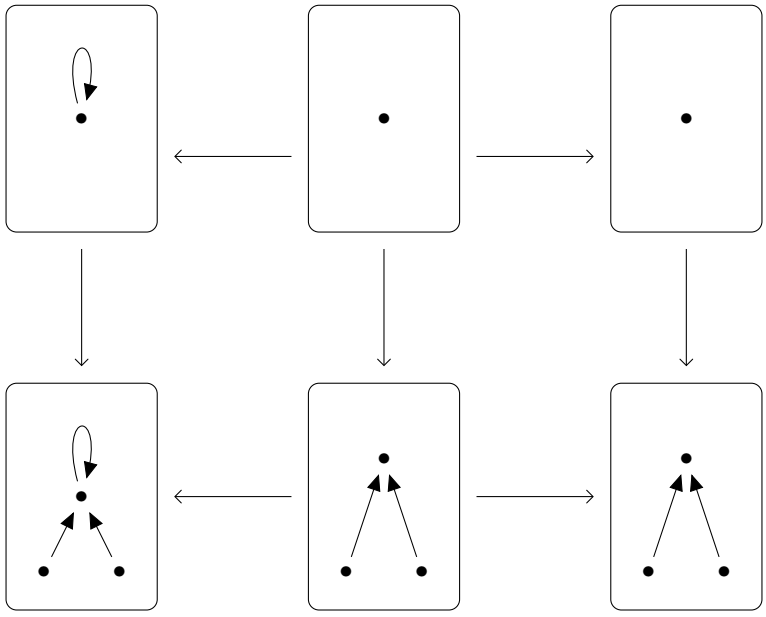
Definition 2.2 (Derived rule)**.** A **derived rule** is any span $g \leftarrow d \rightarrow h$ fitting into the bottom row of the double pushout diagram

$$\begin{array}{ccccc} \ell & \longleftarrow & k & \longrightarrow & r \\ \downarrow & & \downarrow & & \downarrow \\ g & \longleftarrow & d & \longrightarrow & h \end{array}$$

When the arrows of the rule $\ell \leftarrow k \rightarrow r$ are both monic, the arrows of the span $g \leftarrow d \rightarrow h$ are also monic because pushouts preserve monics in topoi [**?**, Lem. 12]. The intuition of this diagram is that $\ell \rightarrow g$ identifies a copy of ℓ in g and we replace that copy with r, resulting in a new object h.

To illustrate, let us return to a system modeled with graphs and where self-loops are meaningless. Then we can apply Rule (1) to any node with a loop. This application is captured with the double pushout diagram

(3)

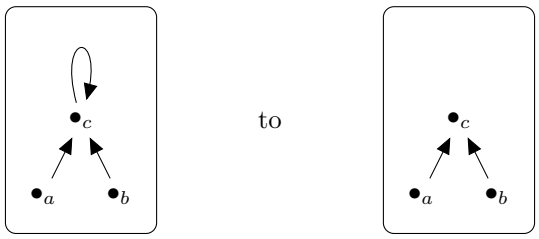


We identified a self-loop in the bottom left graph then applied the rule to remove it. The result is the bottom right graph. The reader can check that the two squares are pushouts.

Usually when modeling a system, there is a set of rewrite rules that accompany it. For example, in resistor circuits there are parallel, series, and star rules. Just like in natural languages, we call a collection of rules a grammar.

Definition 2.3 (Grammar)**.** A topos T together with a finite set P of rules $\{\ell_j \leftarrow k_j \rightarrow r_j\}$ in T is a **grammar**. When the all rules in a grammar have monic arrows, we say the grammar is **fine**. Else, the grammar is **bold**. An arrow of (fine, bold) grammars $(\S, P) \rightarrow (\mathsf{T}, Q)$ is a pullback and pushout preserving functor $F\colon \S \rightarrow \mathsf{T}$ such that for each rule $\ell \xleftarrow{f} k \xrightarrow{g} r$ in P, the rule $F\ell \xleftarrow{Ff} Fk \xrightarrow{Fg} Fr$ is in Q. Together these form a category Gram.

A grammar is a seed. Like a seed, the grammar gives birth to something entirely new and more complex called the language. It is this language that we are interested more so than the grammar. We can certainly learn about the language from the grammar, but what we actually study is the 'rewrite relation' which informs us about how different components of the language relate. Every grammar (T, P) gives rise to a relation $\rightsquigarrow$ on the objects of T defined by $g \rightsquigarrow h$ whenever there exists a rule $g \leftarrow d \rightarrow h$ derived from a production in P. For instance, the above double pushout diagram would relate



But $\rightsquigarrow$ is too small to capture the full behavior of the language. For one, it is not true in general that $g \rightsquigarrow g$ holds. Also, $\rightsquigarrow$ does not capture multi-step rewrites. That is, there may be derived rules witnessing $g \rightsquigarrow g'$ and $g' \rightsquigarrow g''$ but not a derived rule witnessing $g \rightsquigarrow g''$. We want to relate a pair of objects if one can be rewritten into another with a finite sequence of derived rules. Therefore, we actually want the following.

Definition 2.4 (Rewrite relation)**.** To each grammar (T, P), assign a relation on the objects of T defined by setting $g \rightsquigarrow h$ whenever there is a rewrite rule $\ell \leftarrow k \rightarrow r$ in P and an object d of T that fit into a double pushout diagram

$$\begin{array}{ccccc} \ell & \longleftarrow & k & \longrightarrow & r \\ \downarrow & & \downarrow & & \downarrow \\ g & \longleftarrow & d & \longrightarrow & h \end{array}$$

The **rewrite relation** $\rightsquigarrow^*$ is the transitive and reflexive closure of $\rightsquigarrow$.

Every grammar determines a unique rewrite relation in a functorial way. We devote Section **??** to proving this fact, though, we restrict ourselves working with grammars of structured cospan categories.

2.2. **Expressiveness of underlying discrete grammars.** (Show the equivalence of two classes of grammars: discrete and arbitrary. Borrow some language from the original algebraic graph rewriting paper about the classes of grammars.)

As mentioned above, we want to decompose closed systems into open systems. We did not yet mention which open systems are available to use. This depends on context. That is, whatever type of system one has, there is an appropriate grammar stipulated by a theory that describes that system. To illustrate, for an electrical system, a corresponding grammar would have rules for adding resistors in series, or adding the reciprocal of resistors in parallel. Therefore, our starting data is a grammar (X, P)—a topos X and a set of fine rewrite rules $P := \{\ell_j \leftarrow k_j \rightarrow r_j\}$—plus a closed system x in X. Eventually entering the story is a topos A of input

types and an adjunction between A and X. For now, however, we focus on the set of rewrite rules P.

We can prove the main result of this section, Theorem 2.15, by controlling the form of the rewrite rules. In particular, we want the intermediary of the rules, the k_j's, to be 'discrete'. In what follows, we discuss what we mean by 'discrete' and show that the grammar obtained by discretizing (X, P) is just as expressive as (X, P), by which we mean that the induced rewriting relations are equal. This result generalizes a characterization of discrete *graph grammars* given by Ehrig, et. al. [2, Prop. 3.3].

Our concept of 'discreteness' is borrowed from the flat modality on a local topos. However, we avoid the lengthy detour required to discuss the 'flat modality' and a 'local topos'. The background does not add to our story, so we point curious readers elsewhere [4, Ch. C3.6]. By avoiding that detour, we instead require the concept of a comonad, which we present in Definition **??**.

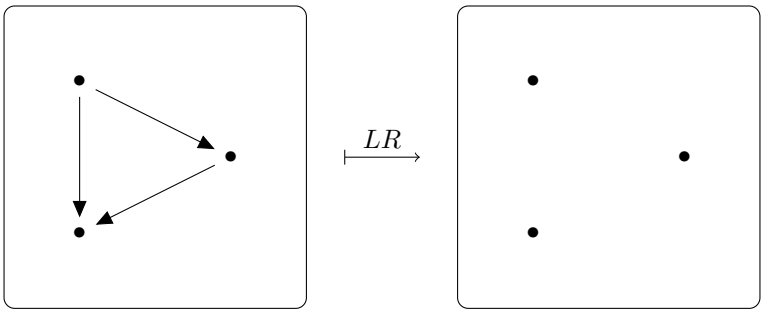
To start our discussion on discreteness, we define a 'discrete comonad'. The definition is straightforward enough, but its purpose may seem alien at first. After the definition, we explain its role in rewriting structured cospans.

Definition 2.5 (Discrete comonad)**.** A comonad on a topos is called **discrete** if its counit is monic. We use $\flat$ to denote a discrete comonad.

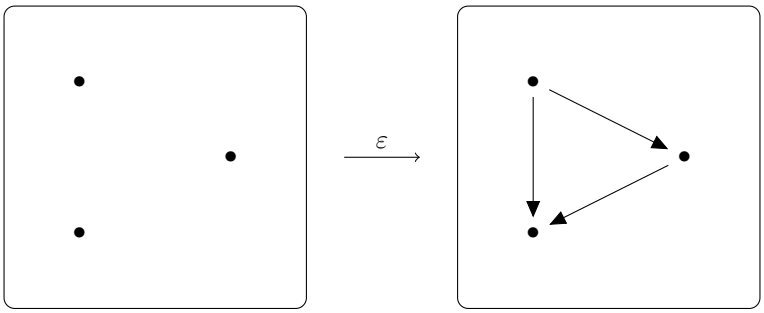
Secretly, we have been working with a discrete comonad all along. The adjunction

$$\mathsf{Set} \underset{L}{\overset{\mathsf{RGraph}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} R\ 4$$

induces the comonad LR on RGraph. Applying LR to a graph x returns the edgeless graph underlying x, hence the term 'discrete'. For example



The counit $\varepsilon_x \colon LRx \to x$ of the comonad LR includes the underlying edgeless graph LRx into the original graph x. For example



Abstractly, this inclusion is why we ask for the counit to be monic. The property we capture with a discrete comonad comes from the systems interpretation of the adjunctions

$$\mathsf{A} \underset{L}{\overset{\mathsf{x}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} R\ 2$$

between topoi. That is, R takes a system x, identifies the largest sub-system that can serve as an interface and turns that sub-system into an interface type Rx. Then L takes that interface type and turns it back into a system LRx. This process effectively strips away every part of a system leaving only those parts that can connect to the outside world. That means LRx is a part of x or, in the parlance of category theory, LRx is a subobject of x. Hence, we ask for a monic counit.

How do we plan to use discrete comonads? We use them to control the form of our grammars. In general, a rewrite rule has form

$$\ell \leftarrow k \rightarrow r$$

where there are no restrictions on what k can be. However, recall that k identifies the part of ℓ that is fixed throughout the rewrite. It does not direct how the rewrite is performed. Therefore, we can deform it a bit without changing the outcome of the applying the rewrite. In particular, we can discretize it by replacing k with $\flat k$. And because $\flat$ has a monic counit, we can insert $\flat k$ right into the middle of the fine rewrite rule.

Definition 2.6 (Discrete grammar)**.** Given a grammar (X, P), define the set $P_\flat$ as consisting of the rules

$$\ell \leftarrow k \leftarrow \flat k \rightarrow k \rightarrow r$$

for each rule $\ell \leftarrow k \rightarrow r$ in P. We call $(X, P_\flat)$ the **discrete grammar** underlying (X, P).

Discrete grammars are easier to work with than arbitrary grammars. So when given an opportunity to work with a discrete grammar instead of a non-discrete grammar, we should take it. Theorem 2.15 gives a sufficient condition that allows us to swap (X, P) for $(\mathsf{X}, P_\flat)$ without consequence. To prove this, however, we borrow from lattice theory which requires that we make a brief turn to fill in some required background.

Definition 2.7 (Lattice)**.** A lattice is a poset $(S, \leq)$ equipped with all finite joins $\bigvee$ and all finite meets $\bigwedge$. It follows that there is a minimal element and maximal element, realized as the empty meet and join respectively, which we denote by 0 and 1.

Joins and meets are also known as suprema and infima. We are using the definition of a lattice common in the category theory literature. This leaves out objects that some mathematicians might consider lattices. Below we give one counter-example and several examples of lattices, the last one being the most relevant.

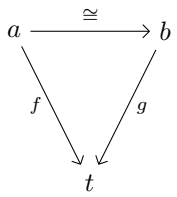
Example 2.8 (Integer Lattice)**.** The integers with the usual ordering $\leq$ do not form a lattice because there is no minimal or maximal element.

Example 2.9 (Lattice of power sets)**.** For any set S, its powerset $\mathcal{P}S$ is a poset via subset inclusion. The powerset becomes a lattice by taking join to be union $a \vee b \coloneqq a \cup b$, and meet to be intersection $a \wedge b \coloneqq a \cap b$. In general, union and

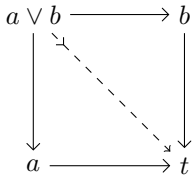
intersection are defined over arbitrary sets, thus realizing arbitrary joins $\bigvee a_\alpha$ and arbitrary meets $\bigwedge a_\alpha$.

Those few examples provide intuition about lattices, but the next example is the most important lattice for us. It is the mechanism by which the power set is generalized into topos theory. It is called the subobject lattice.

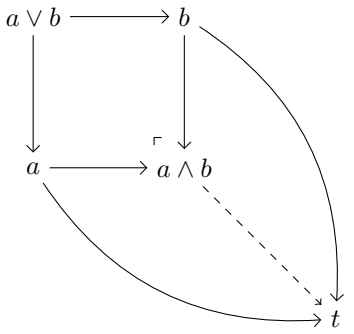
Example 2.10 (Subobject lattice)**.** Let T be a topos and t be an object. There is a lattice $\mathrm{Sub}(t)$ called the subobject lattice of t. The elements of $\mathrm{Sub}(t)$ are called subobjects. They are isomorphism classes of monomorphisms into t. Here, two monomorphisms f, g into t are isomorphic if there is a commuting diagram



The order on $\mathrm{Sub}(t)$ is given by $f \leq g$ if f factors through g, meaning there is an arrow $h\colon a \to b$ such that $f = gh$. Note that h is necessarily monic. The meet operation in $\mathrm{Sub}(t)$ is given by pullback



and join is given by pushout over the meet



We use subobject lattices to characterize which grammars are as expressive as their underlying discrete grammars. To do this, we require subobject lattices with arbitrary meets. The powerset lattice mentioned above has this property, but when do subobject lattices have this property? Here are several sufficient conditions, starting with a well-known result coming from the domain of order theory.

Proposition 2.11. *Any lattice that has all joins also has all meets.*

Proof. Consider a subset S of a lattice. Define the meet of S to be the join of the set of all lower bounds of S. □

Proposition 2.12. *Consider a topos* T *and object* t*. The subobject lattice* $\mathrm{Sub}(t)$ *has arbitrary meets when the over category* $T \downarrow t$ *has all products.*

Proof. Because $T \downarrow t$ is a topos, it has equalizers. Thus giving it all products ensures the existence of all limits, hence meets. □

Corollary 2.13. *Consider a topos* T *and object* t*. The subobject lattice* $\mathrm{Sub}(t)$ *has arbitrary meets when the over category* $T \downarrow t$ *has all coproducts.*

Proof. Combine Propositions 2.11 and 2.12. □

Corollary 2.14. *Consider a presheaf category* $\mathsf{Set}^{\mathsf{C}^{op}}$ *on a small category* C*. For any presheaf* x*,* $\mathrm{Sub}(x)$ *has all meets.*

Proof. The category $\mathsf{Set}^{\mathsf{C}^{\mathrm{op}}} \downarrow x$ of presheaves over x is again a presheaf category by Theorem **??** so has all products. □

At last, we combine the discrete comonad, the discrete grammar, and the complete subobject lattice into a result on the expressiveness on discrete grammars.

Theorem 2.15. *Let* T *be a topos and* $\flat \colon \mathsf{T} \to \mathsf{T}$ *be a discrete comonad. Let* (T, P) *be a grammar such that for every rule* $\ell \leftarrow k \to r$ *in* P*, the subobject lattice* $\mathrm{Sub}(k)$ *has all meets. Then the rewriting relation for* (T, P) *equals the rewriting relation for the underlying discrete grammar* $(\mathsf{T}, P_\flat)$*.*

Proof. Suppose that (T, P) induces $g \rightsquigarrow h$. That means there exists a rule $\ell \leftarrow k \to r$ in P and a derivation

$$\begin{array}{ccccc} \ell & \longleftarrow & k & \longrightarrow & r \\ \downarrow & \lrcorner & \downarrow & \llcorner & \downarrow \\ g & \longleftarrow & d & \longrightarrow & h \end{array} \tag{4}$$

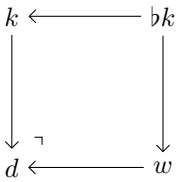
we can achieve that same derivation using rules in $P_\flat$. This requires we build a pushout complement w of the diagram

$$\begin{array}{ccc} k & \xleftarrow{\varepsilon} & \flat k \\ \downarrow & & \\ d & & \end{array}$$

Define

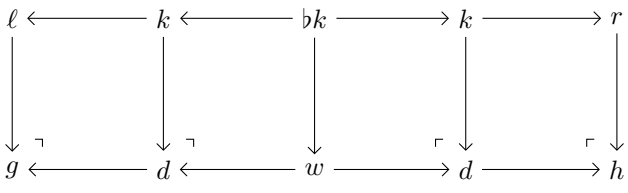
$$w \coloneqq \bigwedge \{z \colon z \vee k = d\} \vee \flat k,$$

This comes with inclusions $\flat k \to w$ and $w \to d$. This w exists because $\mathrm{Sub}(k)$ has all meets. Note that $w \vee k = d$ and $w \wedge k = \flat k$ which means that



is a pushout. It follows that there is a derivation

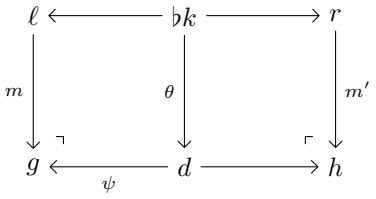
(5)



with respect to $P_\flat$ because, the top row is a rule in $P_\flat$. Therefore, $g \rightsquigarrow h$ via P in Diagram (4) implies that $g \rightsquigarrow^* h$ via $P_\flat$ as shown in Diagram (5).

For the other direction, suppose $g \rightsquigarrow h$ via $P_\flat$, giving a derivation

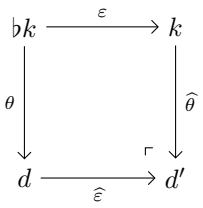
(6)



By construction of $P_\flat$, the rule $\ell \leftarrow \flat k \rightarrow r$ in $P_\flat$ was induced from a rule

$$\ell \xleftarrow{\tau} k \rightarrow r$$

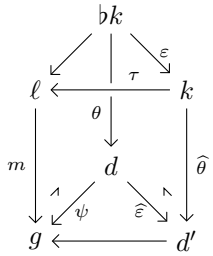
in P, meaning that the map $\flat k \rightarrow \ell$ factors through τ. Next, define d' to be the pushout of the diagram



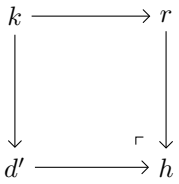
By invoking the universal property of this pushout with the maps

$$\psi \colon d \rightarrow g \quad \text{and} \quad m\tau \colon k \rightarrow \ell \rightarrow g,$$

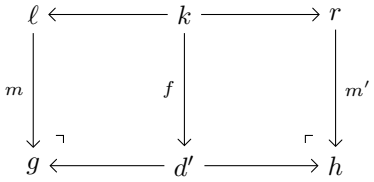
we get a canonical map $d' \rightarrow g$ that we can fit into a commuting diagram



whose back faces are pushouts. Using a standard diagram chasing argument, we can show that the front face is also a pushout. Similarly, the square



is a pushout. Sticking these two pushouts together



shows that $g \rightsquigarrow h$ arises from P.

Because the relation $\rightsquigarrow$ is the same for P and $P_\flat$, it follows that $\rightsquigarrow^*$ is also the same as claimed. □

3. The Structure of Structured Cospans

(Outline
 (a) Strucutred cospas and their morphisms form a topos
 (b) Structured cospans and their rewrites form a double category
The first is a new result here and the section comes from my work with Kenny.)

3.1. **Structured cospans form a topos.** We start this section by recalling and motivating the definition of a structured cospan. As the goal of this paper is to introduce a double pushout (DPO) rewriting theory of structured cospans, we should ensure that the structured cospans have the correct structure to do so. For us, the correct structure is an adhesive category, which Lack and Sobociński [5] defined as a general space in which the desirable properties of rewriting hold. However, our larger goal here is not to maximize generality, but rather to maximize familiarity with the ingredients and so we construct our structured cospans so that they form a topos, which is a special case of an adhesive category [6].

Baez and Courser introduced structured cospans [1] to serve as a syntax for open systems. In order to form a topos out of these, we require stronger conditions than they gave.

Definition 3.1. Fix an adjunction

$$\mathsf{X} \underset{R}{\overset{L}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{A}$$

between (elementary) topoi with L preserving finite limits (that is, a geometric morphism). An **L-structured cospan**, or simply **structured cospan**, is any cospan in X of the form $La \to x \leftarrow Lb$.

A structured cospan is designed to serve as syntax for a system, like a passive linear circuit or Petri net or combinatorial graph, with the ability to connect to compatible systems. We call systems like this *open systems*. The apex of the cospan, the X-object x, represents the system itself. The legs of the cospan, $La \to x$ and $Lb \to x$, select the inputs La and outputs Lb of x. These provide the structured cospans with the compositional property held by those open systems for which we designed this syntax. Namely, two compatible structured cospans, $La \to x \leftarrow Lb$ and $Lb \to y \leftarrow Lc$, can be combined to form the *composite* structured cospan $La \to x +_{Lb} y \leftarrow Lc$, where $x +_{Lb} y$ is obtained by glueing x to y along their common interface Lb. Indeed, we call $La \to x +_{Lb} y \leftarrow Lc$ the composite structured cospan because structured cospans are arrows in a category.

Definition 3.2. Given an adjunction

$$\mathsf{X} \underset{R}{\overset{L}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{A}$$

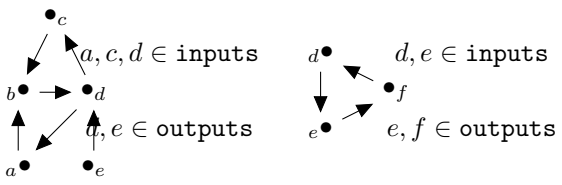
between (elementary) topoi with L preserving finite limits, denote by ${}_L\mathsf{Csp}$ the category whose objects are those of A and arrows from a to b are structured cospans $La \to x \leftarrow Lb$.

Of course, ${}_L\mathsf{Csp}$ does not require the full strength of those conditions to form a category—Baez and Courser do not ask this much—but we choose to work under assumptions that are sufficiently strong to support rewriting and familiar and succint as working with topoi.

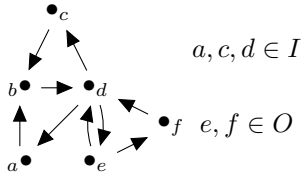
Example 3.3. Consider the adjunction

$$\mathsf{Graph} \underset{R}{\overset{L}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{Set}$$

with L the discrete graph functor and R the functor that forgets the graph edges. Structured cospans here are open graphs, that is graphs equipped with two subset of nodes, one serves as inputs and another, which is not necessarily disjoint, serves as outputs.



Connect these open graphs by gluing like-nodes together. This results in



In Lack and Sobociński axiomatisation of DPO rewriting, rewriting is done on the *objects* of a category, not the arrows. Therefore, to rewrite structured cospans, we require a category where the structured cospans are objects and so we are obliged to define morphisms of structured cospans. The expected definition is given.

Definition 3.4. A morphism from $La \to x \leftarrow Lb$ to $Lc \to y \leftarrow Ld$ is a triple of arrows (f, g, h) fitting into the commuting diagram

$$\begin{array}{ccccc} La & & \longrightarrow & & Lb \\ {\scriptstyle Lf}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle Lh} \\ Lc & & \longrightarrow & & Ld \end{array}$$

There is a category, denoted by ${}_L\mathsf{StrCsp}$, whose objects are L-structured cospans and whose arrows are their morphisms.

At this point, certain readers may have observed that structured cospans fit as objects into one category and as arrows into another, thus making it likely they fit into a double category. Indeed, they do. In fact, they fit into a symmetric monoidal double category which Baez and Courser discuss. We, however, are uninterested in this for our purposes.

We now come to the first of our main results: that ${}_L\mathsf{StrCsp}$ is a topos. This result is critical for our theory because it allows the introduction of rewriting onto structured cospans.

Theorem 3.5. *Given an adjunction*

$$\mathsf{X} \underset{R}{\overset{L}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{A}$$

between topoi with L preserving finite limits, the category ${}_L\mathsf{StrCsp}$ is a topos.

Proof. By adjointness, ${}_L\mathsf{StrCsp}$ is equivalent to the category whose objects are cospans of form $a \to Rx \leftarrow b$ and morphisms are triples (f, g, h) fitting into the commuting diagram

$$\begin{array}{ccccc} w & & \longrightarrow & & Ra & & \longleftarrow & & x \\ {\scriptstyle f}\downarrow & & & & {\scriptstyle Rg}\downarrow & & & & \downarrow{\scriptstyle h} \\ y & & \longrightarrow & & Rb & & \longleftarrow & & z \end{array}$$

This, in turn, is equivalent to the comma category $(\mathsf{A}\times\mathsf{A} \downarrow \Delta R)$ where $\Delta\colon \mathsf{A} \to \mathsf{A}\times\mathsf{A}$ is the diagonal functor, a right adjoint. Hence, ΔR is a right adjoint making $(\mathsf{A} \times \mathsf{A} \downarrow \Delta R)$ an Artin glueing [7], therefore a topos. □

Knowing that ${}_L\mathsf{StrCsp}$ is a topos is enough for us introduce rewriting structured cospans, but by pausing for a moment, we see that ${}_L\mathsf{StrCsp}$ is constructed functorially.

Theorem 3.6. *Denote by* Topos *the category whose objects are topoi and whose arrows from* X *to* A *are adjoint pairs*

$$\mathsf{X} \underset{R}{\overset{L}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathsf{A}$$

where L *preserves finite limits (these arrows are geometric morphisms, a standard morphism between elementary topoi). There is a functor*

$$_{(-)}\mathsf{StrCsp}_{(-)} \colon [\bullet \to \bullet, \mathsf{Topos}] \to \mathsf{Topos}$$

defined by

$$\begin{array}{ccc} \mathsf{X} & \overset{L}{\underset{R}{\overset{\longleftarrow}{\bot}}} & \mathsf{A} \\ {\scriptstyle F}\big\downarrow\!\dashv & & \dashv\big\downarrow{\scriptstyle F'} \\ \mathsf{X}' & \underset{L'}{\overset{R'}{\underset{\longrightarrow}{\overset{\longleftarrow}{\top}}}} & \mathsf{A}' \end{array} \quad \overset{\mathsf{StrCsp}_{(-)}}{\longmapsto} \quad \begin{array}{ccc} & \overset{\Theta}{\longrightarrow} & \\ {}_{L}\mathsf{StrCsp} & \bot & {}_{L'}\mathsf{StrCsp} \\ & \underset{\Theta'}{\longleftarrow} & \end{array}$$

which is in turn given by

$$\begin{array}{ccccc} La & \xrightarrow{m} & x & \xrightarrow{n} & Lb \\ {\scriptstyle Lf}\big\downarrow & & {\scriptstyle g}\big\downarrow & & \big\downarrow{\scriptstyle Lh} \\ Lc & \xrightarrow{o} & y & \xrightarrow{p} & Ld \end{array} \quad \overset{\Theta}{\longmapsto} \quad \begin{array}{ccccc} L'G'a & \xrightarrow{Gm} & Gx & \xleftarrow{Gn} & L'G'b \\ {\scriptstyle L'G'f}\big\downarrow & & {\scriptstyle Gg}\big\downarrow & & \big\downarrow{\scriptstyle L'G'h} \\ L'G'c & \xrightarrow{Go} & Gy & \xleftarrow{Gp} & L'G'd \end{array}$$

and

$$\begin{array}{ccccc} L'a' & \xrightarrow{m'} & x' & \xleftarrow{n'} & L'b' \\ {\scriptstyle L'f'}\big\downarrow & & {\scriptstyle g'}\big\downarrow & & \big\downarrow{\scriptstyle L'h'} \\ L'c' & \xrightarrow{o'} & y' & \xleftarrow{p'} & L'd' \end{array} \quad \overset{\Theta'}{\longmapsto} \quad \begin{array}{ccccc} LF'a' & \xrightarrow{Fm'} & Fx' & \xleftarrow{Fn'} & LF'b' \\ {\scriptstyle LF'f'}\big\downarrow & & {\scriptstyle Fg'}\big\downarrow & & \big\downarrow{\scriptstyle LF'h'} \\ LF'c' & \xrightarrow{Fo'} & Fy' & \xleftarrow{Fp'} & LF'd' \end{array}$$

Proof. In light of Theorem 3.5, it suffices to show that $\Theta \dashv \Theta'$ gives a geometric morphism.

Denote the structured cospans

$$[m, n] \colon La \to x \leftarrow Lb$$

in ${}_{L}\mathsf{StrCsp}$ by ℓ and

$$[m', n'] \colon L'a' \to x' \leftarrow L'b'$$

in ${}_{L'}\mathsf{StrCsp}$ by ℓ'. Denote the unit and counit for $F \dashv G$ by η, ε and for $F' \dashv G'$ by η', ε'. The assignments

$$\begin{aligned} ((f, g, h) \colon \ell \to \Theta'\ell') &\mapsto ((\varepsilon' \circ F'f, \varepsilon \circ Fg, \varepsilon' \circ F'h) \colon \Theta\ell \to \ell') \\ ((f', g', h') \colon \Theta\ell \to \ell') &\mapsto ((G'f' \circ \eta', Gg' \circ \eta, G'h' \circ \eta') \colon \ell \to \Theta'\ell') \end{aligned}$$

give a bijection $\hom(\Theta\ell, \ell') \simeq \hom(\ell, \Theta'\ell')$. Moreover, it is natural in ℓ and ℓ'. This rests on the natural maps η, ε, η', and ε'. The left adjoint Θ' preserves finite limits because they are taken pointwise and L, F, and F' all preserve finite limits. □

With this theorem, we have brought forth various categories of the form ${}_L\mathsf{StrCsp}$ where L runs through finite limit preserving left adjoints. For reference, we refer to such categories as structured cospan categories. This leaves us wanting to organize structured cospan categories into some kind of structure, say a 2-category. This is easily accomplished using the following as our 1-arrows.

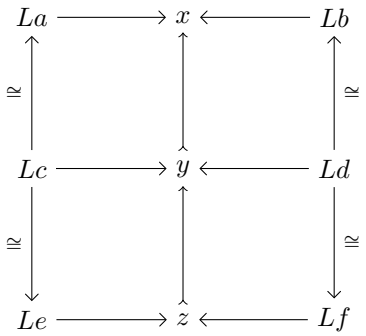
Definition 3.7. A **functor of structured cospan categories** is a pair of finitely continuous and cocontinuous functors $F\colon \mathsf{X} \to \mathsf{X}'$ and $G\colon \mathsf{A} \to \mathsf{A}'$ such that $FL = L'F$ and $GR = R'F$.

Structured cospan categories along with their functors and natural transformations form a 2-category **StrCsp**. However, we leave this 2-category here and more onwards towards our intended goal: rewriting structured cospans.

3.2. **Structured cospan rewrites for a double category.** (Cover what a grammar is and organize them into a double category a la FineRewrite. Uncertain to include: interchange law, symmetric monoidal structure. Just point to thesis? Dno't call things FineRewrite, just Rewrite.)

In this chapter, we introduce a theory of fine rewriting to structured cospans. Rewriting is *fine* when the rewrite rules are spans with monic legs. Our primary goal is to define a double category whose squares are fine rewrites of structured cospans. The rough idea is that this double category, denoted ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$, has interface types for objects, structured cospans for horizontal arrows, isomorphisms of interface objects for vertical arrows, and fine rewrite rules of structured cospans for squares. We prove in Proposition 3.12 that ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$ actually is a double category. The first step to proving this is to ensure the fine rewrite rules are suitable squares for our double category, we define them as follows.

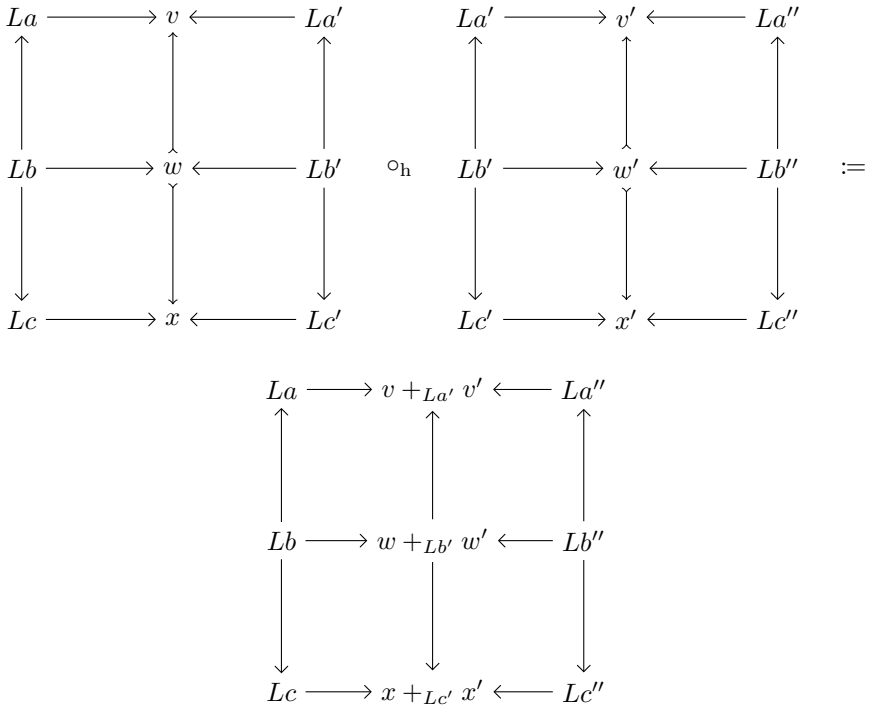
Definition 3.8 (Fine rewrite)**.** A **fine rewrite of structured cospans** is an isomorphism class of spans of structured cospans of the form



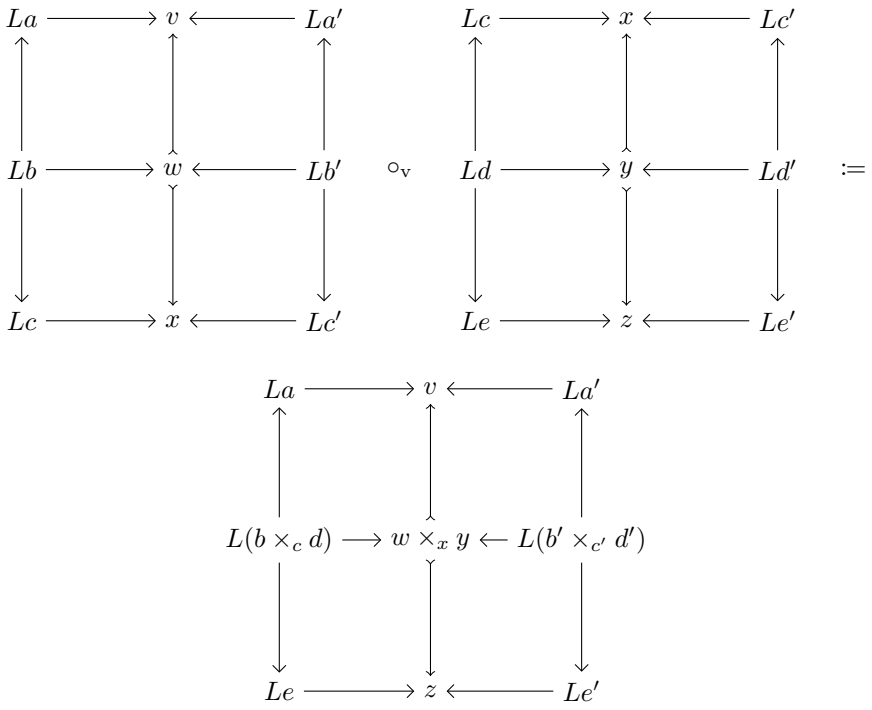
The marked arrows are monic.

In a double category, the squares have two composition operations. Horizontal composition uses pushout as is typical with cospan categories. The vertical composition uses pullback as is typical in span categories. But because there are no higher order arrows traversing the squares in a double category, and because pushouts and pullbacks are only defined up to isomorphism, we take isomorphism classes of structured cospan rewrite rules. With the squares of ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$ defined, we can introduce the two composition operations.

Definition 3.9. The **horizontal composition** of fine rewrite rules is given by



The **vertical composition** of fine rewrite rules is



We defined $\circ_{\mathrm{h}}$ and $\circ_{\mathrm{v}}$ using representatives of isomorphism classes, however this operation is well-defined. It is less clear, however, that these operations preserve the monic arrows in the fine rewrites of structured cospans. In Proposition 3.11, we show that horizontal and vertical composition do preserve these monic arrows. To prove this, we require the following lemma.

Lemma 3.10. *The diagram*

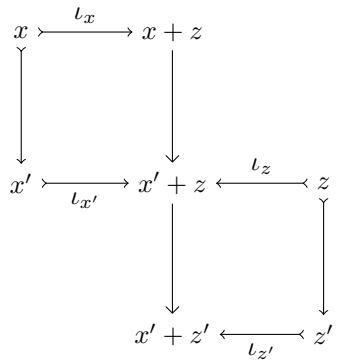
$$\begin{array}{ccccc} x & \longleftarrow & y & \longrightarrow & z \\ \downarrow & & \downarrow\cong & & \downarrow \\ x' & \longleftarrow & y' & \longrightarrow & z' \end{array} \tag{7}$$

induces a pushout

$$\begin{array}{ccc} x+z & \xrightarrow{\rho} & x+_y z \\ {\scriptstyle\gamma}\downarrow & & \downarrow{\scriptstyle\gamma'} \\ x'+z' & \xrightarrow[\rho']{} & x'+_{y'} z \end{array} \tag{8}$$

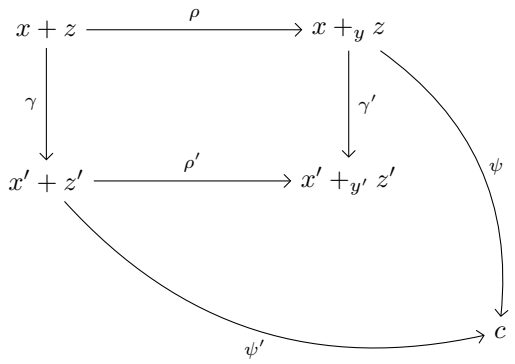
such that the canonical arrows γ and γ' are monic.

Proof. The universal property of coproducts implies that γ factors through $x'+z$ as in the diagram



It is straightforward to check that both squares are pushouts. By Lemma **??**, it follows that γ is monic.

Diagram 8 commutes because of the universal property of coproducts. To see that it is a pushout, arrange a cocone



(9)

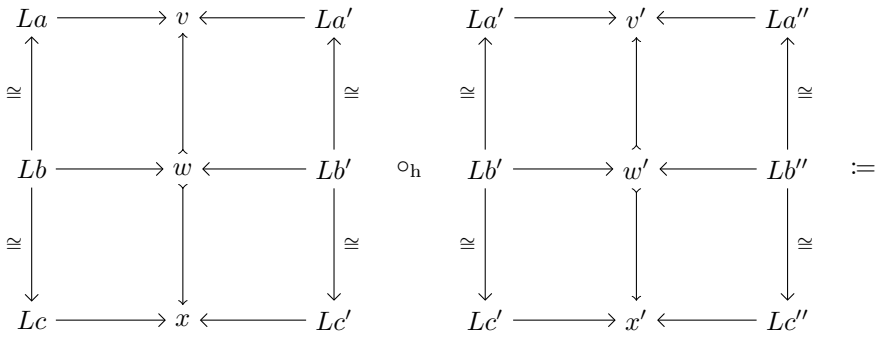
Denote by ι_x any map that includes x. Then $\psi'\iota_{x'}$, $\psi'\iota_{z'}$, and c form a cocone under the span $x' \leftarrow y' \rightarrow z'$ from the bottom face of Diagram 7. This induces the canonical map $\psi'' \colon x' +_{y'} z' \rightarrow c$. It follows that $\psi'\iota_{x'} = \psi''\rho'\iota_{x'}$ and $\psi'\iota_{z'} = \psi''\rho'\iota_{z'}$. Therefore $\psi' = \psi''\rho'$ by the universal property of coproducts.

Furthermore, $\psi\rho\iota_z$, $\psi\rho\iota_z$, and c form a cocone under the span $x \leftarrow y \rightarrow z$ on the top face of Diagram 7. then $\psi\rho\iota_x = \psi'\gamma\iota_x = \psi''\rho'\gamma\iota_x = \psi''\psi'\rho\iota_x$ and $\psi\rho\iota_z = \psi'\gamma\iota_z = \psi''\rho'\gamma\iota_z = \psi''\gamma'\rho\iota_z$ meaning that both ψ and $\psi''\psi'$ satisfy the canonical map $x +_y z \rightarrow d$. Hence $\psi = \psi''\psi'$.

The universality of ψ'' with respect to Diagram 9 follows from the universality of γ'' with respect to $x' +_{y'} z'$. □

Lemma 3.11. *Horizontal and vertical composition of fine rewrites are fine rewrites.*

Proof. We can see that the span of cospan obtained by horizontal composition of fine rewrites



$$\begin{array}{ccccc}
La & \longrightarrow & v +_{La'} v' & \longleftarrow & La'' \\
\cong\uparrow & & \uparrow & & \uparrow\cong \\
Lb & \longrightarrow & w +_{Lb'} w' & \longleftarrow & Lb'' \\
\cong\downarrow & & \downarrow & & \downarrow\cong \\
Lc & \longrightarrow & x +_{Lc'} x' & \longleftarrow & Lc''
\end{array}$$

is again a fine rewrite, that is the arrows $w +_{Le} x \to u +_{Lb} v$ and $w +_{Le} x \to y +_{Lh} z$ are monic, by applying Lemma 3.10 to the diagrams

$$\begin{array}{ccccc}
v & \longleftarrow & La' & \longrightarrow & v' \\
\uparrow & & \cong\uparrow & & \uparrow \\
w & \longleftarrow & Lb' & \longrightarrow & w'
\end{array}
\qquad \text{and} \qquad
\begin{array}{ccccc}
w & \longleftarrow & Lb' & \longrightarrow & w' \\
\downarrow & & \cong\downarrow & & \downarrow \\
x & \longleftarrow & Lc' & \longrightarrow & x'
\end{array}$$

The result for vertical composition

$$\begin{array}{ccccc}
La & \longrightarrow & v & \longleftarrow & La' \\
\cong\uparrow & & \uparrow & & \uparrow\cong \\
Lb & \longrightarrow & w & \longleftarrow & Lb' \\
\cong\downarrow & & \downarrow & & \downarrow\cong \\
Lc & \longrightarrow & x & \longleftarrow & Lc'
\end{array}
\circ_{\mathrm{v}}
\begin{array}{ccccc}
Lc & \longrightarrow & x & \longleftarrow & Lc' \\
\cong\uparrow & & \uparrow & & \uparrow\cong \\
Ld & \longrightarrow & y & \longleftarrow & Ld' \\
\cong\downarrow & & \downarrow & & \downarrow\cong \\
Le & \longrightarrow & z & \longleftarrow & Le'
\end{array}
\coloneqq
\begin{array}{ccccc}
La & \longrightarrow & v & \longleftarrow & La' \\
\cong\uparrow & & \uparrow & & \uparrow\cong \\
L(b \times_c d) & \longrightarrow & w \times_x y & \longleftarrow & L(b' \times_{c'} d') \\
\cong\downarrow & & \downarrow & & \downarrow\cong \\
Le & \longrightarrow & z & \longleftarrow & Le'
\end{array}$$

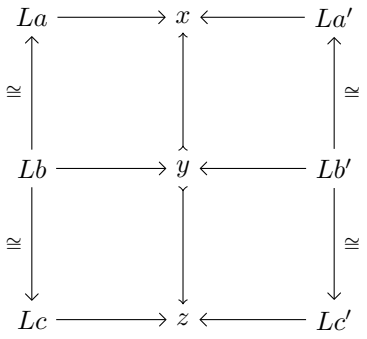
holds because pullback preserves monomorphisms. □

With horizontal and vertical composition in hand, we construct the double category ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$. Actually, we delay discussing the interchange law until Section **??** because it is difficult enough to warrant its own section.

Proposition 3.12. *Let*

$$\mathsf{A} \underset{L}{\overset{\mathsf{X}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} R\ 2$$

be a adjunction with L preserving pullbacks. There is a double category ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$ *whose objects are the* A*-objects, horizontal arrows of type* $a \to b$ *are structured cospans* $La \to x \leftarrow Lb$*, vertical arrows are spans in* A *with invertible arrows, and squares are fine rewrites of structured cospans*



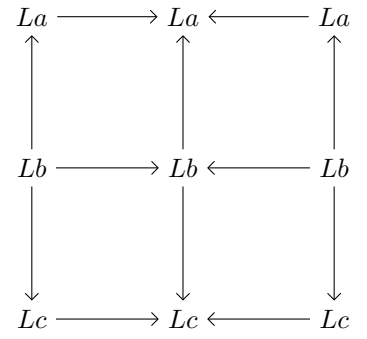
Proof. This proof requires we check the axioms of a double category as laid out in Definition **??**. For simplicity, we denote ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$ by $\mathbb{R}$ in this proof.

The object category $\mathbb{R}_0$ is given by objects of A and isomorphism classes of spans in A such that each leg is an isomorphism. The arrow category $\mathbb{R}_1$ has as objects the structured cospans

$$La \to x \leftarrow La'$$

and as morphisms the fine rewrites of structured cospans.

The functor $U \colon \mathbb{R}_0 \to \mathbb{R}_1$ acts on objects by mapping a to the identity cospan on La and on morphisms by mapping $La \leftarrow Lb \to Lc$, whose legs are isomorphisms, to the square



The functor $S\colon \mathbb{R}_1 \to \mathbb{R}_0$ acts on objects by sending $La \to x \leftarrow La'$ to a and on morphisms by sending a square

$$\begin{array}{ccccc} La & \longrightarrow & x & \longleftarrow & La' \\ \uparrow & & \uparrow & & \uparrow \\ Lb & \longrightarrow & y & \longleftarrow & Lb' \\ \downarrow & & \downarrow & & \downarrow \\ Lc & \longrightarrow & z & \longleftarrow & Lc' \end{array}$$

to the span $La \leftarrow Lb \to Lc$. The functor T is defined similarly sends an object

$$La \to x \leftarrow La'$$

of $\mathbb{R}_1$ to a' a square

$$\begin{array}{ccccc} La & \longrightarrow & x & \longleftarrow & La' \\ \uparrow & & \uparrow & & \uparrow \\ Lb & \longrightarrow & y & \longleftarrow & Lb' \\ \downarrow & & \downarrow & & \downarrow \\ Lc & \longrightarrow & z & \longleftarrow & Lc' \end{array}$$

to the span $La' \leftarrow Lb' \to Lc'$.

The horizontal composition functor

$$\odot\colon \mathbb{R}_1 \times_{\mathbb{R}_0} \mathbb{R}_1 \to \mathbb{R}_1$$

acts on objects by composing cospans with pushouts in the usual way. It acts on morphisms by

$$\begin{array}{ccccccc} La & \longrightarrow & v & \longleftarrow & La' & \longrightarrow & v' & \longleftarrow & La'' \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ Lb & \longrightarrow & w & \longleftarrow & Lb' & \longrightarrow & w' & \longleftarrow & Lb'' \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ Lc & \longrightarrow & x & \longleftarrow & Lc' & \longrightarrow & x' & \longleftarrow & Lc'' \end{array} \quad \overset{\odot}{\longmapsto} \quad \begin{array}{ccccc} La & \longrightarrow & v +_{La'} v' & \longleftarrow & La'' \\ \uparrow & & \uparrow & & \uparrow \\ Lb & \longrightarrow & w +_{Lb'} w' & \longleftarrow & Lb'' \\ \downarrow & & \downarrow & & \downarrow \\ Lc & \longrightarrow & x +_{Lc'} x' & \longleftarrow & Lc'' \end{array}$$

Section **??** is devoted to proving that $\odot$ is functorial, that is, it preserves composition. It is straightforward to check that the required equations are satisfied. The

associator and unitors are given by natural isomorphisms that arise from universal properties. □

And now, our double category of fine rewrites is defined. It remains to prove the interchange law, which we do next.

4. The language of structured cospans

(Outline
(a) Introduce grammars and language of structured cospans.
(b) Show that the Language functor characterizes the rewriting relation on a grammar of structured cospans.
)

The idea of decomposing a whole into parts has long been useful. It exists across human endeavors, be it academic, artistic, or artisanal. A biologist decomposes life into genuses and species. A literary critic decomposed a play into acts and scenes. A sommelier decomposes a wine into color, viscosity, aroma, and taste. In this chapter, as do the biologist, critic, and sommelier, we too decompose. Though for us, we decompose a closed system into open sub-systems.

Given that this paper fits into a larger wave of research into applying category theory to compositional systems, we place decomposition into this context. A strong motivation in this wave of research is to build a toolbox with which to study complex systems by decomposing them into simpler components and studying those instead. In a truly compositional system, information gleaned from the components can be aggregated to give infomation about the original system.

As mathematicians, we must bring rigor to our decomposition. In this chapter, we do just that. We start by formalizing closed systems as structured cospans with an empty interface $0 \to x \leftarrow 0$. Then, using the fine rewriting paradigm from Chapter **??**, we place structured cospans into the double category ${}_L\mathbb{F}\mathbf{ine}\mathbb{R}\mathbf{ewrite}$ as horizontal 1-arrows. To decompose a closed system

$$L0 \to x \leftarrow L0$$

is to write an arrow as a composite of arrows

$$L0 \to x_1 \leftarrow La_1 \to x_2 \leftarrow La_2 \cdots La_{n-1} \to x_n \leftarrow L0$$

We use such decompositions to prove our main result which states that two structured cospans

$$L0 \to x \leftarrow L0 \quad \text{and} \quad L0 \to x' \leftarrow L0$$

are equivalent precisely when there is a square between them. We interpret this result in three ways.

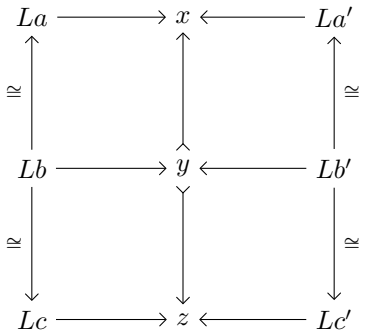
(a) It shows that the rewriting relation for a closed system is functorial and is characterized using squares in a double category.
(b) A closed system decomposes into open systems, and simplifying each open system simplifies the composite closed system.
(c) Open systems provide a local perspective on the closed perspective via this decomposition.

There are two main thrusts to this proof. The first generalizes a classification of formal graph grammars given by Ehrig, et. al. [2]. This is Theorem 2.15. Gadducci and Heckel proved this in the case of graphs [3], but our result generalizes this to structured cospans. Our proof mirrors theirs.

4.1. **Rewriting structured cospans.** (Generalize gadducci/heckle's theorem. Place it into the ACT context.)

Equipped with knowledge about when grammars and their underlying discrete grammars generate the same rewriting relation, we continue towards goal of decomposing closed systems. First, we revisit Section **??** to get some facts about grammars. We then obtain the language associated to a grammar in a functorial way. Finally, we show how to decompose into open subsystems a given system equipped with a grammar.

Recall the category Gram. The objects of Gram are pairs (T, P) where T is a topos and P is a set of rewrite rules in T. The arrows $(\mathsf{T}, P) \to (\mathsf{T}', P')$ of Gram are rule-preserving functors $\mathsf{T} \to \mathsf{T}'$. Our interest now lies in the full subcategory of structured cospan grammars $\mathsf{StrCspGram}$ whose objects are the grammars of form $({}_L\mathsf{StrCsp}, P)$ where P consists of fine rewrites of structured cospans, meaning they have the form



and the left adjoint L has a monic counit.

It is on this category $\mathsf{StrCspGram}$ that we define a functor encoding the rewrite relation to each grammar. We denote this functor

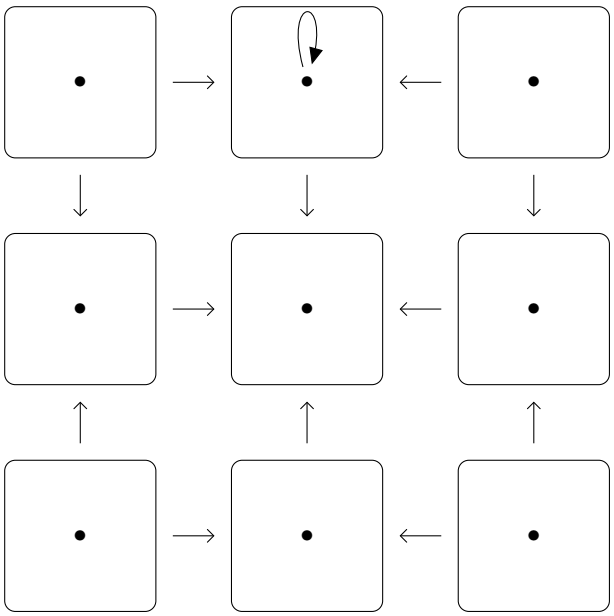
$$\mathrm{Lang}\colon \mathsf{StrCspGram} \to \mathsf{DblCat}$$

where Lang is short for 'language'. This is an appropriate term as this functor provides *(i)* the terms formed by connecting together open systems (instead of, in linguistics, concatenating units of syntax) and *(ii)* the rules governing how to interchange open systems (instead of parts of speech). To help visualize this, we sketch a simple example.

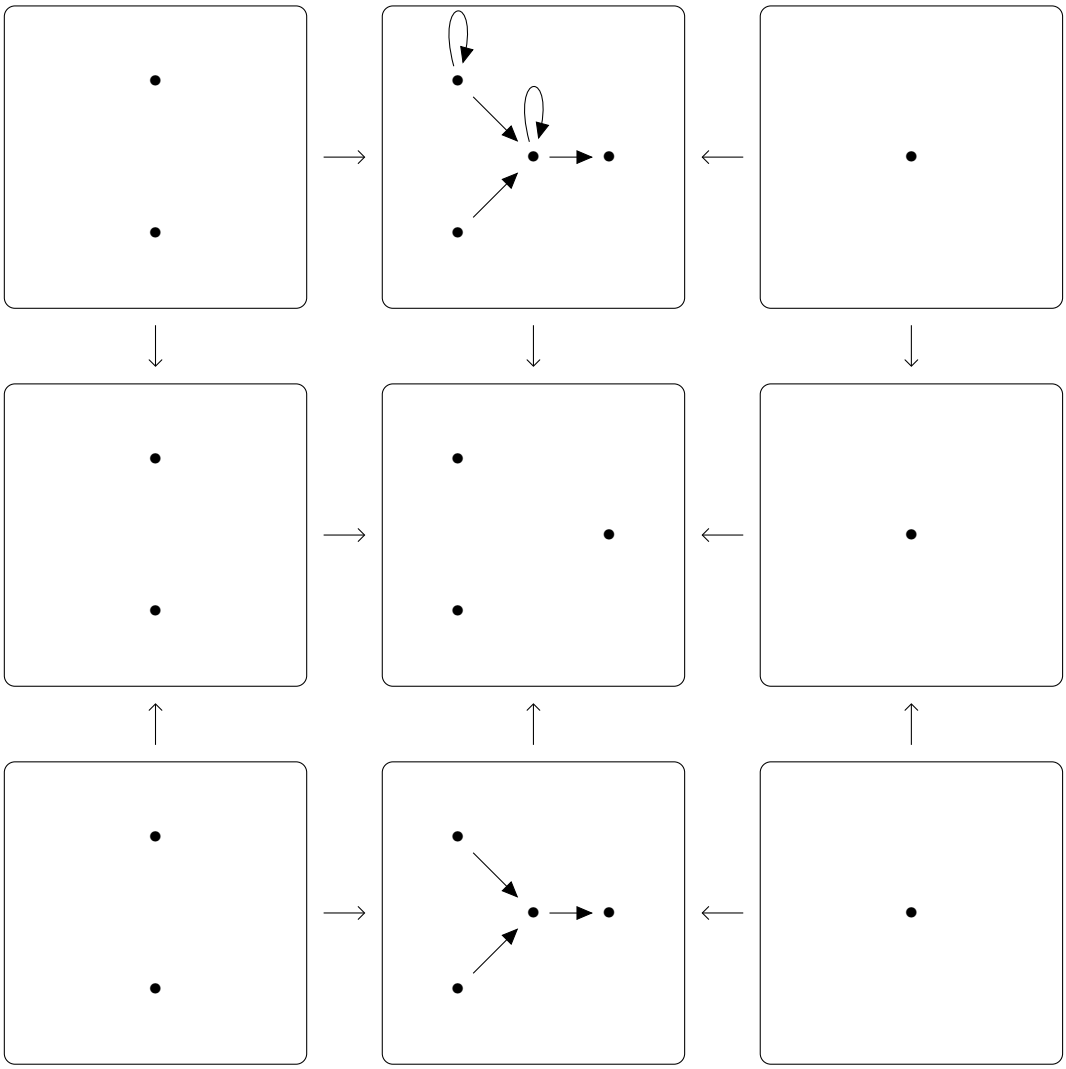
Example 4.1. Start with the, by now familiar, adjunction

$$\mathsf{Set} \underset{L}{\overset{\mathsf{RGraph}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} R\ 4$$

For this L, ${}_L\mathsf{StrCsp}$ is the category of open graphs. Make a grammar from ${}_L\mathsf{StrCsp}$ by defining a P to have the single rule



The language associated to this grammar consists of all open graphs. The rewrite relation says $g \rightsquigarrow^* h$ if we obtain h be removing loops from g. We illustrate this with the following square in the double category $\mathrm{Lang}({}_L\mathsf{StrCsp}, P)$.



To actually construct Lang, we use functors $D\colon \mathsf{StrCspGram} \to \mathsf{StrCspGram}$ and $S\colon \mathsf{StrCspGram} \to \mathsf{DblCat}$. Roughly, D sends a grammar $({}_L\mathsf{StrCsp}, P)$ to all of the rewrite rules derived from P and S generates a double category on the squares obtained from the rewrite rules of a grammar $({}_L\mathsf{StrCsp}, P)$. In this way, we get the language of a grammar as a double category where the squares are the rewrite rules. The next lemma defines D and gives some of its properties.

Lemma 4.2. *There is an idempotent functor $D\colon \mathsf{StrCspGram} \to \mathsf{StrCspGram}$ defined as follows. On objects define $D({}_L\mathsf{StrCsp}, P)$ to be the grammar $({}_L\mathsf{StrCsp}, P_D)$, where P_D consists of all rules $g \leftarrow h \to d$ witnessing the relation $g \rightsquigarrow h$ with respect to $({}_L\mathsf{StrCsp}, P)$. On arrows, define $DF\colon D({}_L\mathsf{StrCsp}, P) \to D({}_{L'}\mathsf{StrCsp}, Q)$ to be F. Moreover, the identity on $\mathsf{StrCspGram}$ is a subfunctor of D.*

Proof. That $D({}_L\mathsf{StrCsp}, P)$ actually gives a grammar follows from the fact that pushouts respect monics in a topos [5, Lem. 12].

To show that D is idempotent, we show that for any grammar $({}_L\mathsf{StrCsp}, P)$, we have $D({}_L\mathsf{StrCsp}, P) = DD({}_L\mathsf{StrCsp}, P)$. Rules in $DD({}_L\mathsf{StrCsp}, P)$ appear in the bottom row of a double pushout diagram whose top row is a rule in $D({}_L\mathsf{StrCsp}, P)$, which in turn is the bottom row of a double pushout diagram whose top row is in $({}_L\mathsf{StrCsp}, P)$. Thus, a rule in $DD({}_L\mathsf{StrCsp}, P)$ is the bottom row of a double pushout diagram whose top row is in $({}_L\mathsf{StrCsp}, P)$. See Figure 1.

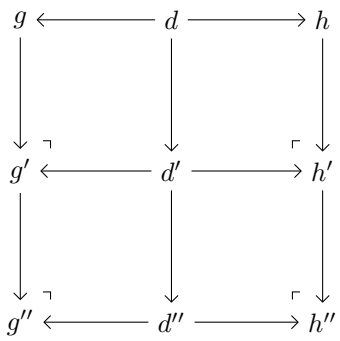


FIGURE 1. Stacked double pushout diagrams

The identity is a subfunctor of D because $\ell \rightsquigarrow r$ for any production $\ell \leftarrow k \rightarrow r$ in $({}_L\mathsf{StrCsp}, P)$ via a triple of identity arrows. Hence there is a monomorphism

$$({}_L\mathsf{StrCsp}, P) \rightarrow D({}_L\mathsf{StrCsp}, P)$$

induced from the identity functor on ${}_L\mathsf{StrCsp}$. □

In this lemma, we have created a functor D that sends a grammar to a new grammar consisting of all derived rules. That D is idempotent means that all rules derived from P can be derived directly; multiple applications of D are unnecessary. That the identity is a subfunctor of D means that set of the derived rules P_D contains the set of initial rules P.

The next stage in defining Lang is to define $S\colon \mathsf{StrCspGram} \rightarrow \mathsf{DblCat}$. On objects, let $S({}_L\mathsf{StrCsp}, P)$ be the sub-double category of ${}_L\mathbb{S}\mathbf{trCsp}$ generated by the rules in P considered as squares. On arrows, S sends

$$F\colon ({}_L\mathsf{StrCsp}, P) \rightarrow ({}_{L'}\mathsf{StrCsp}, P')$$

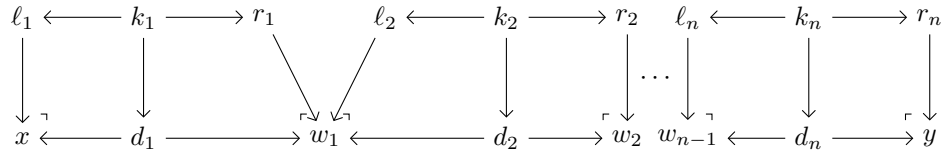
to the double functor defined that extends the mapping between the generators of $S({}_L\mathsf{StrCsp}, P)$ and $S({}_{L'}\mathsf{StrCsp}, P')$. This preserves composition because F preserves pullbacks and pushouts.

Definition 4.3. (Language of a grammar) The **language functor** is defined to be Lang $\coloneqq SD$.

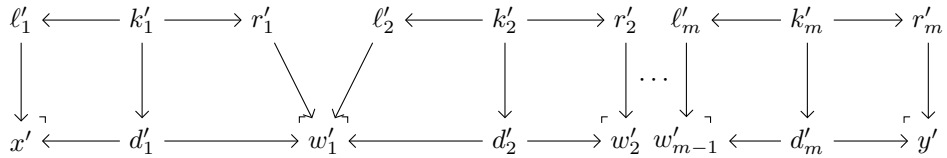
To witness the rewriting relation on a closed system as a square in a double category, we require this next lemma that formalizes the analogy between rewriting the disjoint union of systems and tensoring squares.

Lemma 4.4. *If $x \rightsquigarrow^* y$ and $x' \rightsquigarrow^* y'$, then $x + x' \rightsquigarrow^* y + y'$*

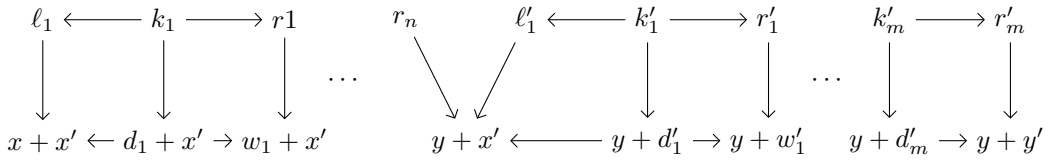
Proof. If the derivation $x \rightsquigarrow^* y$ comes from a string of double pushout diagrams



and the derivation $x' \rightsquigarrow^* y'$ comes from a string of double pushout diagrams



realize $x + x' \rightsquigarrow^* y + y'$ by



□

As promised, we can now decompose closed systems into open systems. For this, we need a topos of closed systems X equipped with a grammar (X, P). The closed systems need interfaces, meaning we need to introduce an adjunction

$$\mathsf{A} \underset{L}{\overset{\mathsf{x}}{\underset{\perp}{\longleftarrow}}} R\,2$$

where L preserves pullbacks and has a monic counit. At this point, the material from the previous section returns. This adjunction gives a discrete comonad $\flat := LR$ from which we form the discrete grammar $(\mathsf{X}, P_\flat)$. Now define the structured cospan grammar $({}_L\mathsf{StrCsp}, \widehat{P_\flat})$ where $\widehat{P_\flat}$ contains the rule

$$\begin{array}{ccccc} L0 & \longrightarrow & \ell & \longleftarrow & LRk \\ \uparrow & & \uparrow & & \uparrow \\ L0 & \longrightarrow & LRk & \longleftarrow & LRk \\ \downarrow & & \downarrow & & \downarrow \\ L0 & \longrightarrow & r & \longleftarrow & LRk \end{array} \tag{10}$$

for each rule $\ell \leftarrow LRk \to r$ of $P_\flat$. We use $({}_L\mathsf{StrCsp}, \widehat{P_\flat})$ to prove our main theorem.

Before stating the theorem, we note that this theorem generalizes work by Gadducci and Heckel [3] whose domain of inquiry was graph rewriting. The arc of our proof follows theirs.

Theorem 4.5. *Fix an adjunction $(L \dashv R)\colon \mathsf{X} \rightleftarrows \mathsf{A}$ with monic counit. Let (X, P) be a grammar such that for every X-object x in the apex of a production of P, the lattice $\mathrm{Sub}(x)$ has all meets. Given g, $h \in \mathsf{X}$, then $g \rightsquigarrow^* h$ in the rewriting relation for a grammar (X, P) if and only if there is a square*

$$\begin{array}{ccccc}
LR0 & \longrightarrow & g & \longleftarrow & LR0 \\
\uparrow & & \uparrow & & \uparrow \\
LR0 & \longrightarrow & d & \longleftarrow & LR0 \\
\downarrow & & \downarrow & & \downarrow \\
LR0 & \longrightarrow & h & \longleftarrow & LR0
\end{array}$$

in the double category $\mathrm{Lang}({}_L\mathsf{StrCsp}, \widehat{P_\flat})$.

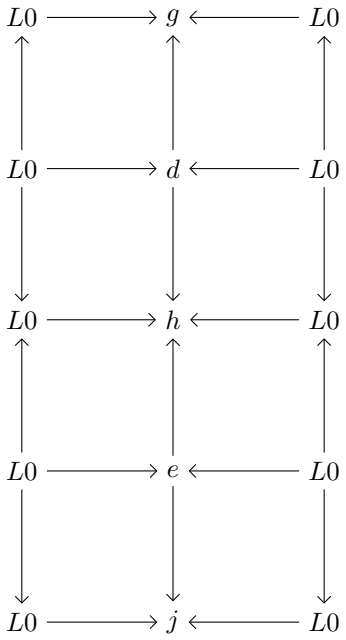
Proof. We show sufficiency by inducting on the length of the derivation. If $g \rightsquigarrow^* h$ in a single step, meaning that there is a diagram

$$\begin{array}{ccccc}
\ell & \longleftarrow & LRk & \longrightarrow & r \\
\downarrow & & \downarrow & & \downarrow \\
g & \longleftarrow & d & \longrightarrow & h
\end{array}$$

then the desired square is the horizontal composition of

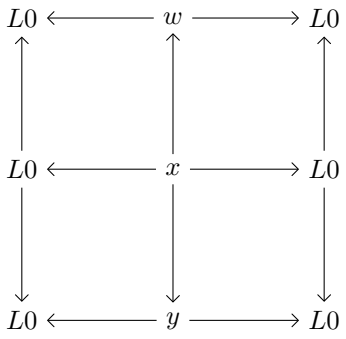
$$\begin{array}{ccccccccc}
L0 & \longrightarrow & \ell & \longleftarrow & LRk & \longrightarrow & d & \longleftarrow & L0 \\
\uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
L0 & \longrightarrow & LRk & \longleftarrow & LRk & \longrightarrow & d & \longleftarrow & L0 \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
L0 & \longrightarrow & r & \longleftarrow & LRk & \longrightarrow & d & \longleftarrow & L0
\end{array}$$

The left square is a generator and the right square is the identity on the horizontal arrow $LRk \to d \leftarrow L0$. The square for a derivation $g \rightsquigarrow^* h \rightsquigarrow j$ is the vertical composition of

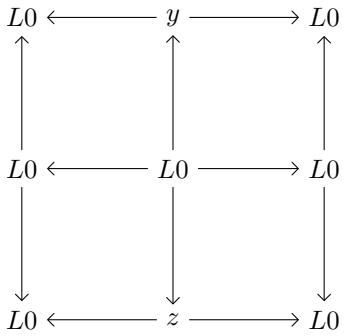


The top square is from $g \rightsquigarrow^* h$ and the second from $h \rightsquigarrow j$.

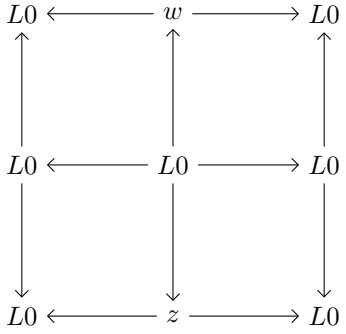
Conversely, proceed by structural induction on the generating squares of $\mathrm{Lang}({}_L\mathsf{StrCsp}, \widehat{P_\flat})$. It suffices to show that the rewrite relation is preserved by vertical and horizontal composition by generating squares. Suppose we have a square



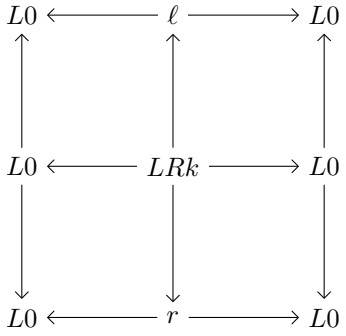
corresponding to a derivation $w \rightsquigarrow^* y$. Composing this vertically with a generating square, which must have form



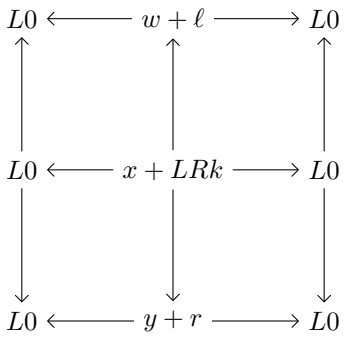
corresponding to a production $y \leftarrow L0 \rightarrow z$ gives



which corresponds to a derivation $w \rightsquigarrow^* y \rightsquigarrow z$. Composing horizontally with a generating square

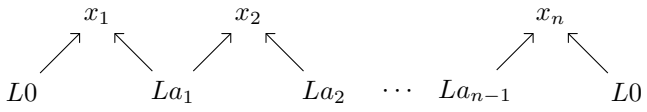


corresponding with a production $\ell \leftarrow LRk \rightarrow r$ results in the square

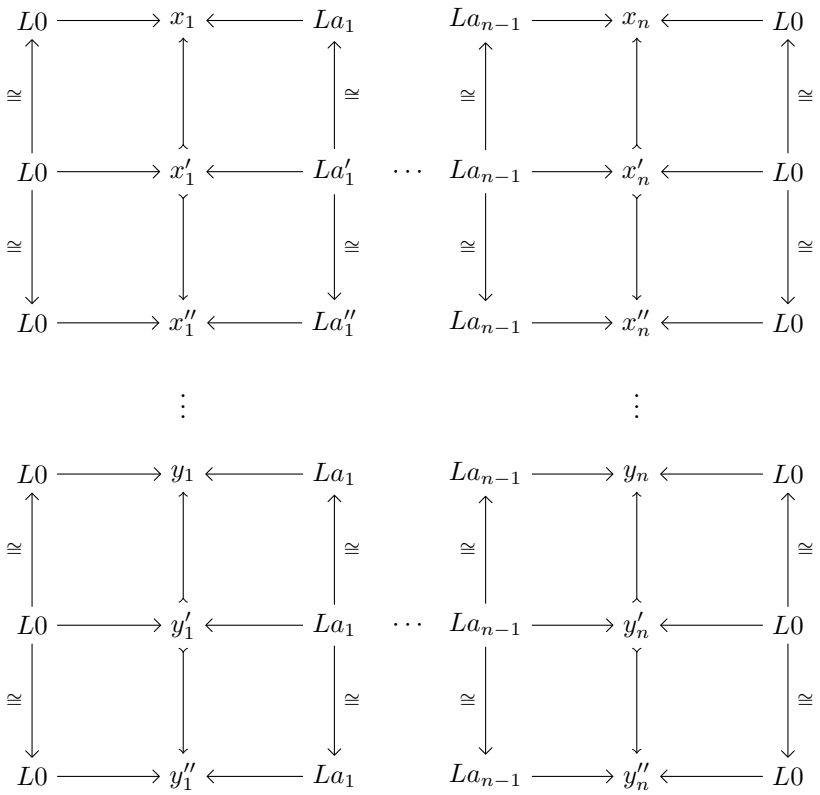


But $w + \ell \rightsquigarrow^* y + r$ as seen in Lemma 4.4. □

With this result, we have completely described the rewrite relation for a grammar (X, P) with squares in $\mathrm{Lang}({}_L\mathsf{StrCsp}, \widehat{P_\flat})$ framed by the initial object of X. These squares are rewrites of a closed system in the sense that the interface is empty. We can instead begin with a closed system x in X as represented by a horizontal arrow $L0 \to x \leftarrow L0$ in $\mathrm{Lang}({}_L\mathsf{StrCsp}, \widehat{P_\flat})$ and decompose it into a composite of sub-systems, that is a sequence of composable horizontal arrows



Rewriting can be performed on each of these sub-systems



The composite of these squares is a rewriting of the original system.

References

[1] John C. Baez and Kenny Courser. Structured cospans. *In Preperation.* (Referred to on page 10.)

[2] H. Ehrig, M. Pfender, and H. J. Schneider. Graph-grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory (Univ. Iowa, Iowa City, Iowa, 1973)*, pages 167–180. 1973. (Referred to on page 5, 21.)

[3] F. Gadducci and R. Heckel. An inductive view of graph transformation. In *Recent trends in algebraic development techniques (Tarquinia, 1997)*, volume 1376 of *Lecture Notes in Comput. Sci.*, pages 223–237. Springer, Berlin, 1998. (Referred to on page 21, 27.)

[4] Peter T Johnstone. *Sketches of an elephant: A topos theory compendium*, volume 2. Oxford University Press, 2002. (Referred to on page 5.)

[5] Stephen Lack and PawełSobociński. Adhesive categories. In *Foundations of software science and computation structures*, volume 2987 of *Lecture Notes in Comput. Sci.*, pages 273–288. Springer, Berlin, 2004. (Referred to on page 10, 25.)

[6] Stephen Lack and PawełSobociński. Toposes are adhesive. In *Graph transformations*, volume 4178 of *Lecture Notes in Comput. Sci.*, pages 184–198. Springer, Berlin, 2006. (Referred to on page 10.)

[7] Gavin Wraith. Artin glueing. *J. Pure Appl. Algebra*, 4:345–348, 1974. (Referred to on page 12.)

Email address: dcicala@newhaven.edu

Department of Mathematics and Physics, University of New Haven