Grammar transformation with DPO rewriting

Aleks Kissinger [1] Vladimir Zamdzhiev [2]

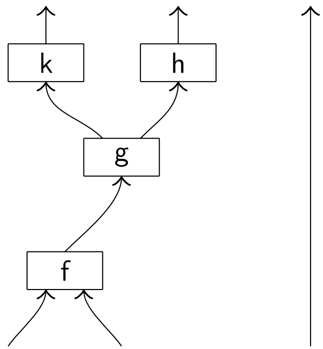
[1]iCIS
Radboud University

[2]Department of Computer Science
University of Oxford

2 April 2016

String Diagrams

Example



- First introduced by Roger Penrose in 1971 as alternative to the tensor-index notation used in theoretical physics.
- (Typed) nodes connected via (typed) wires
- Wires do not have to be connected to nodes at either end
- Open-ended wires serve as inputs/outputs
- Emphasis on compositionality

String diagram applications

Applications in:

- Monoidal category theory (sound and complete categorical reasoning)

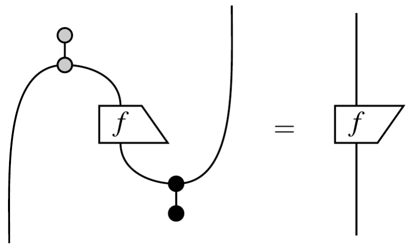


Figure: J. Vicary, W. Zeng (2014)

- Quantum computation and information (graphical calculi, e.g. ZX-calculus)

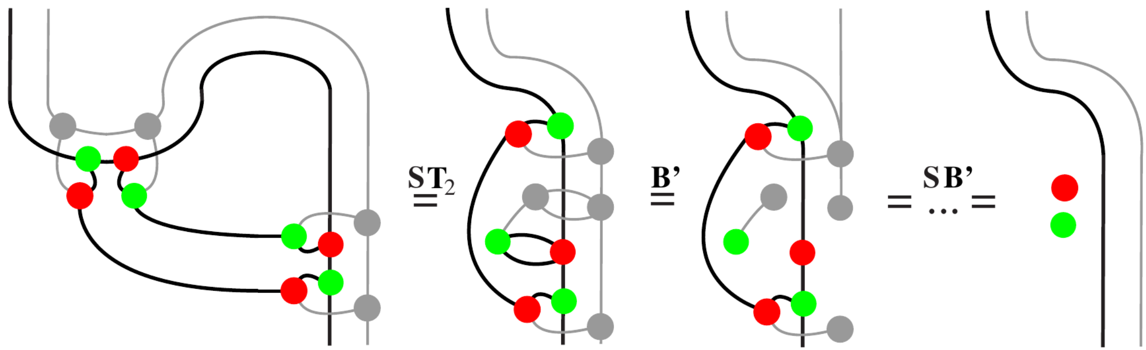


Figure: B. Coecke, R. Duncan (2011)

String diagram applications

- Concurrency (Petri nets)

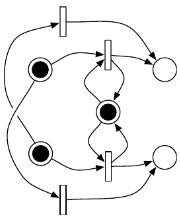


Figure: P. Sobocinski (2010)

- Computational linguistics (compositional semantics)

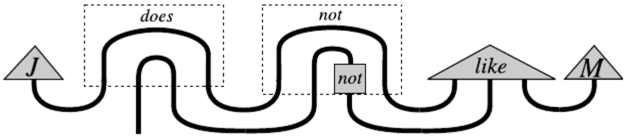
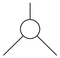


Figure: B. Coecke, E. Grefenstette, M. Sadrzadeh (2013)

String Diagram Example

A monoid is a triple $(A, \cdot, 1)$, such that:

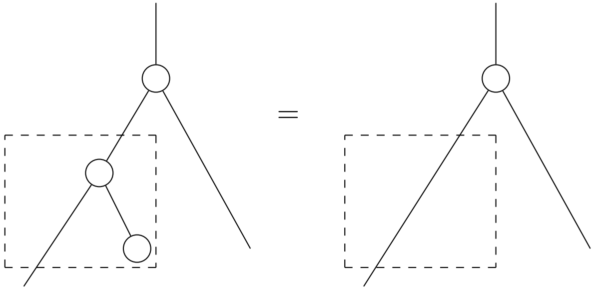
$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \text{and} \quad 1 \cdot a = a = a \cdot 1$$

Setting $(\_ \cdot \_) :=$  and $1 :=$  ,we get:

String Diagram Example

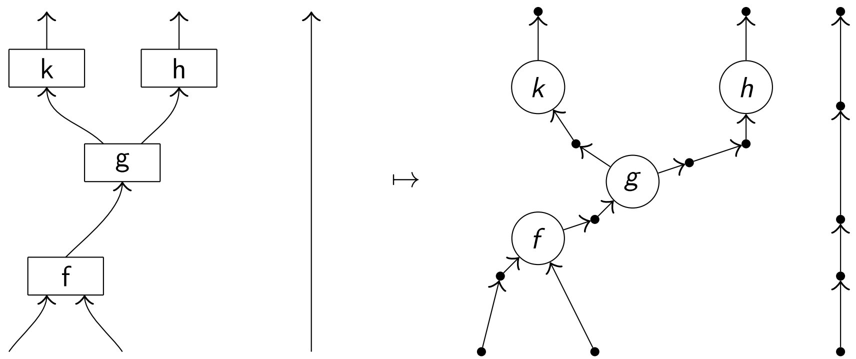
Equational reasoning is performed by replacing subdiagrams:

Example



String Graphs

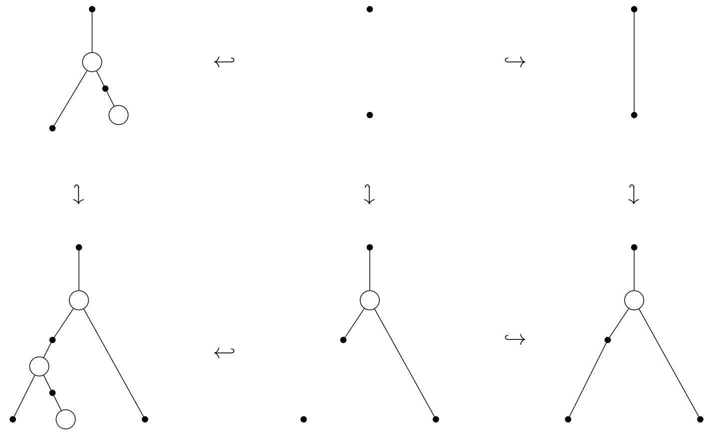
Example



- String diagrams are formally described using (non-discrete) topological notions
- This is problematic for computer implementations
- Discrete representation exists in the form of *String Graphs*
- String graphs are typed (directed) graphs, such that:
  - Every vertex is either a *node-vertex* or a *wire-vertex*
  - No edges between node-vertices
  - In-degree of every wire-vertex is at most one
  - Out-degree of every wire-vertex is at most one

Reasoning with String Graphs

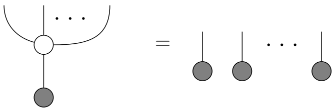
We use double-pushout (DPO) rewriting on string graphs to represent string diagram rewriting:



Families of string diagrams

- String diagrams (and string graphs) can be used to establish equalities between pairs of objects, one at a time.
- Proving infinitely many equalities simultaneously is only possible using metalogical arguments.

Example



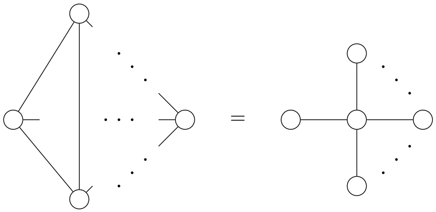
- However, this is imprecise and implementing software support for it would be very difficult.

Motivation

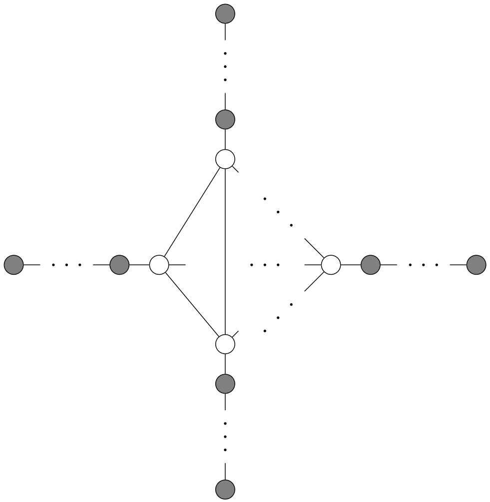
- Given an equational schema between two families of string diagrams, how can we apply it to a target family of string diagrams and obtain a new equational schema?

Example

Equational schema between complete graphs on n vertices and star graphs on n vertices:

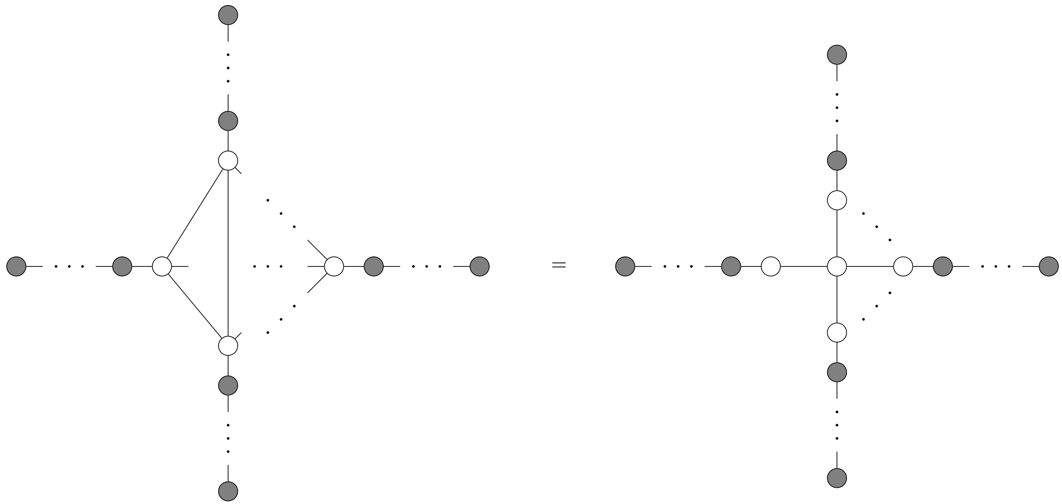


Then, we can apply this schema to the following family of graphs:



Motivation

and we obtain a new equational schema:



The main ideas are:

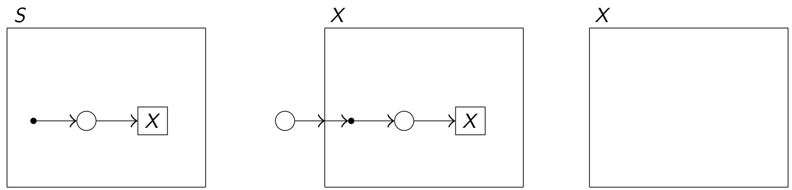
- Context-free graph grammars represent families of graphs
- "Grammar" DPO rewrite rules represent equational schemas
- "Grammar" DPO rewriting represents equational reasoning on families of graphs
- "Grammar" DPO rewriting is admissible (or correct) w.r.t. concrete instantiations

Context-free graph grammars

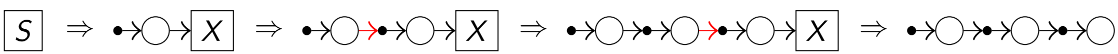
- We investigate context-free graph grammars first, as they have better structural, complexity and decidability properties compared to other more expressive graph grammars.
- Most studied context-free graph grammars are:
  - Hyperedge replacement grammars (HR)
  - Vertex replacement grammars (VR)
- Large body of literature available for both VR and HR grammars
- VR grammars (also known as C-edNCE grammars) are more expressive than HR grammars in general
- We will be working with VR grammars only, in particular boundary grammars (B-edNCE)

edNCE grammar example

The following grammar generates the set of all chains of node vertices with an input and no outputs:



A derivation in the above grammar of the string graph with three node vertices:



where we color the newly established edges in red.

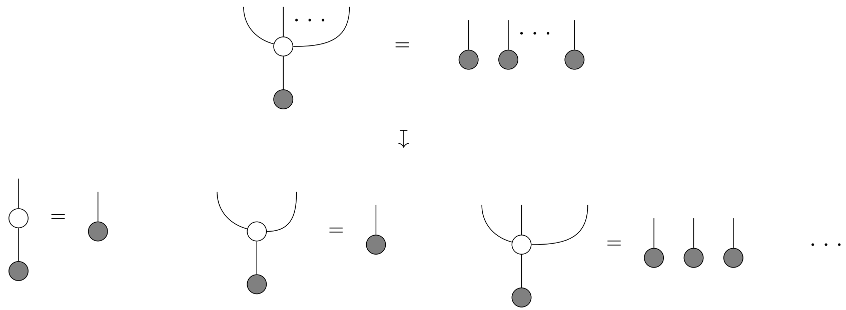
- An edNCE grammar is a graph-like structure – essentially it is a partition of graphs equipped with connection instructions

Adhesivity of edNCE grammars

- The category of (slightly generalized) edNCE grammars **GGram** is an adhesive category
- Suitable for performing DPO rewriting
- DPO rewriting along with gluing conditions in **GGram** are straightforward generalisations of the standard DPO method
- Languages induced by edNCE grammars are defined set-theoretically, not algebraically
- Restrictions on rewrite rules and matchings necessary if we wish rewriting in **GGram** to make sense w.r.t language generation

Quantification over equalities

- an equational schema between two families of string diagrams establishes infinitely many equalities:



- How do we model this using edNCE grammars?
- Idea: DPO rewrite rule in **GGram**, where productions are in 1-1 correspondance

Grammar rewrite pattern

Definition (Grammar rewrite pattern)

A *Grammar rewrite pattern* is a triple of grammars B_L, B_I and B_R, such that there is a bijection between their productions which also preserves non-terminals and their labels.

Definition (Pattern instantiation)

Given a grammar rewrite pattern (B_L, B_I, B_R), a pattern instantiation is given by a triple of concrete derivations:

$$S \Longrightarrow^{B_L}_{v_1,p_1} H_1 \Longrightarrow^{B_L}_{v_2,p_2} H_2 \Longrightarrow^{B_L}_{v3,p3} \cdots \Longrightarrow^{B_L}_{v_n,p_n} H_n$$

and

$$S \Longrightarrow^{B_I}_{v_1,p_1} H'_1 \Longrightarrow^{B_I}_{v_2,p_2} H'_2 \Longrightarrow^{B_I}_{v3,p3} \cdots \Longrightarrow^{B_I}_{v_n,p_n} H'_n$$

and

$$S \Longrightarrow^{B_R}_{v_1,p_1} H''_1 \Longrightarrow^{B_R}_{v_2,p_2} H''_2 \Longrightarrow^{B_R}_{v3,p3} \cdots \Longrightarrow^{B_R}_{v_n,p_n} H''_n$$

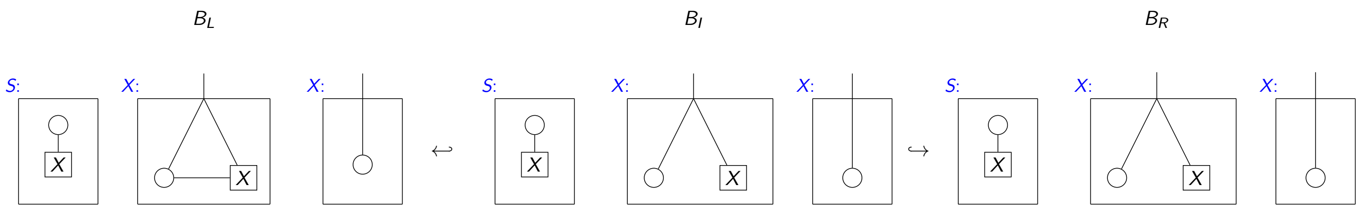
- That is, we always expand the same non-terminals in the three sentential forms in parallel

Theorem

Every pattern instantiation is a DPO rewrite rule on graphs.

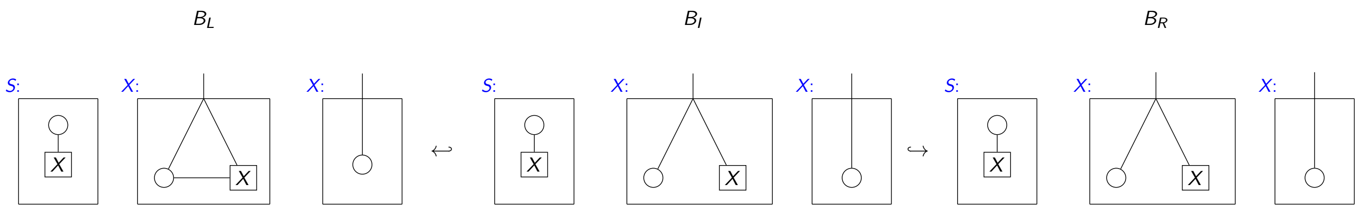
Grammar rewrite pattern

Example



Grammar rewrite pattern

Example



- Instantiation :

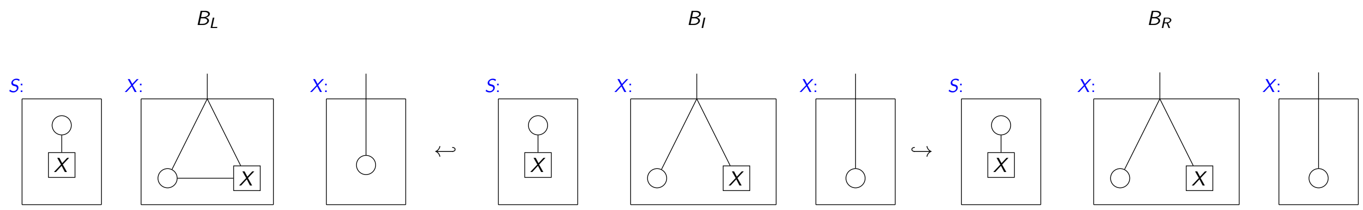
S

S

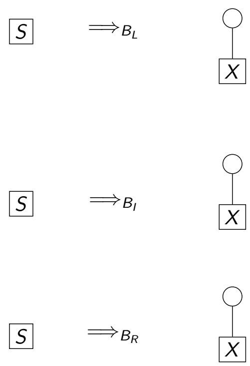
S

Grammar rewrite pattern

Example

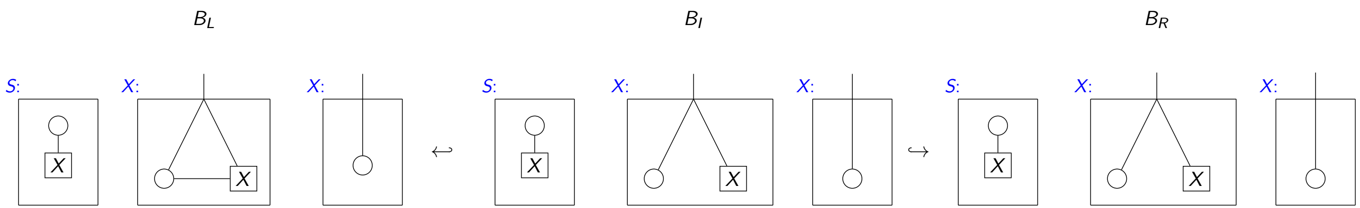


- Instantiation :

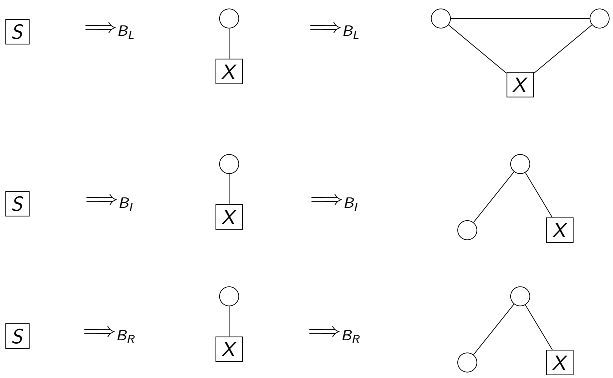


Grammar rewrite pattern

Example

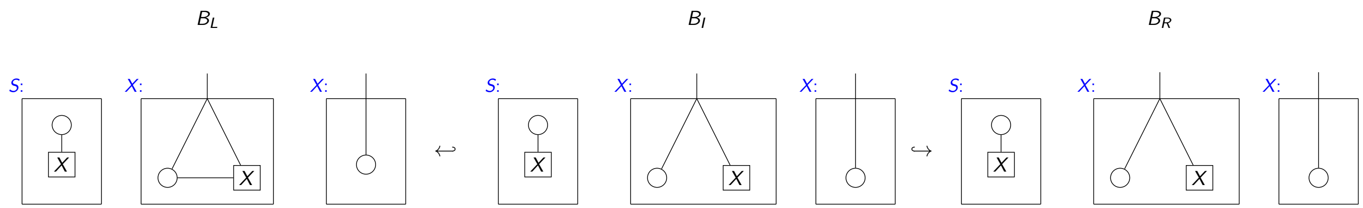


- Instantiation :

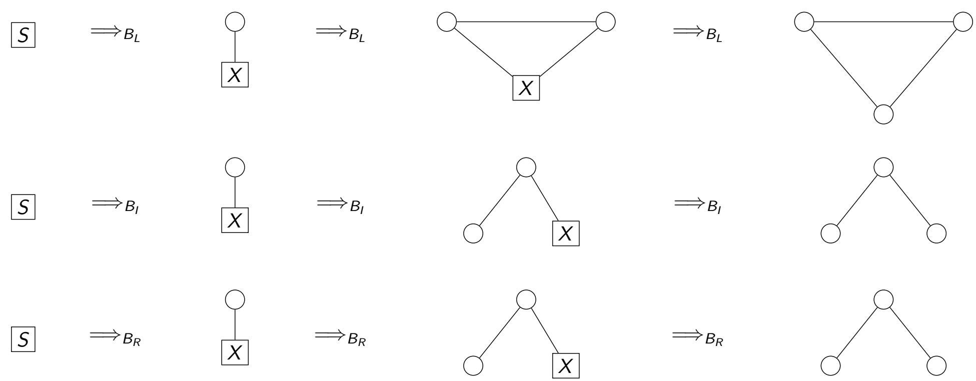


Grammar rewrite pattern

Example

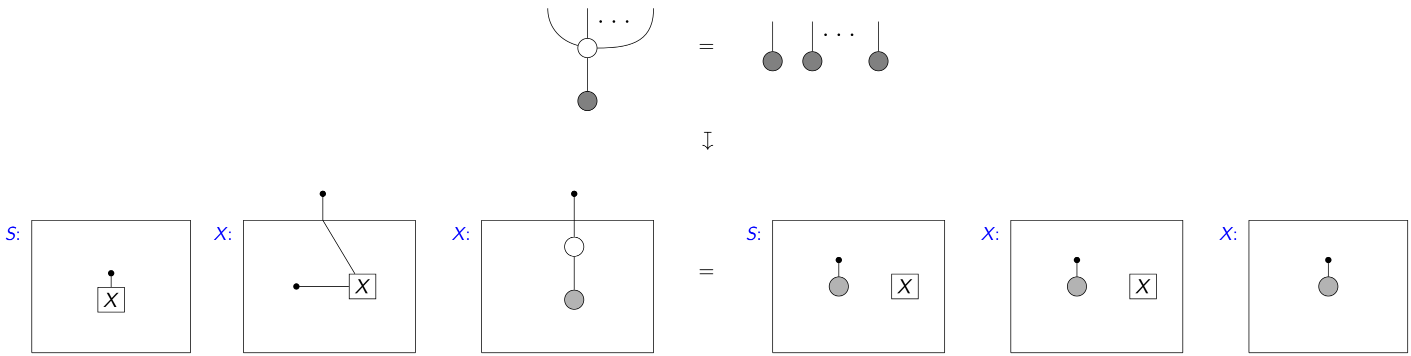


- Instantiation :



Obtaining new equalities

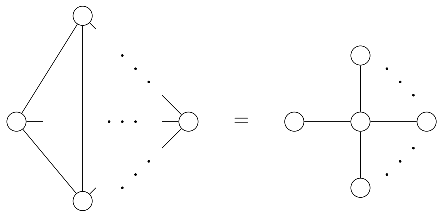
- We can encode infinitely many equalities between string diagrams by using grammar rewrite patterns



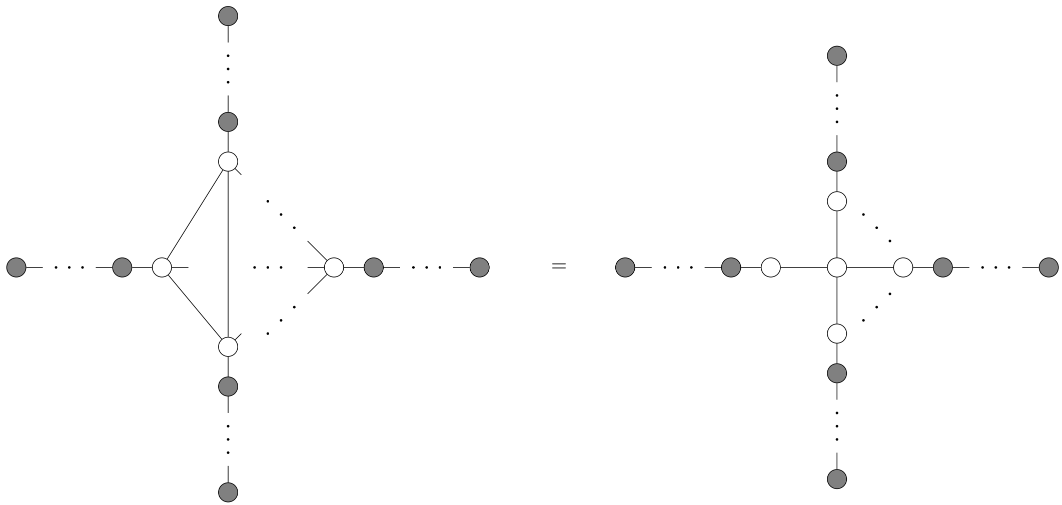
- Next, we show how to rewrite a family of diagrams using an equational schema in an admissible way

Example

Given an equational schema:

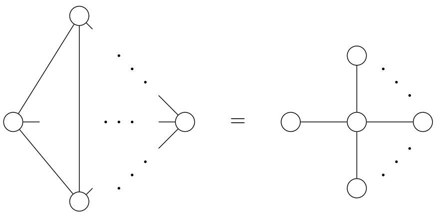


how do we apply it to a target family of string diagrams (left) and get the resulting family (right):

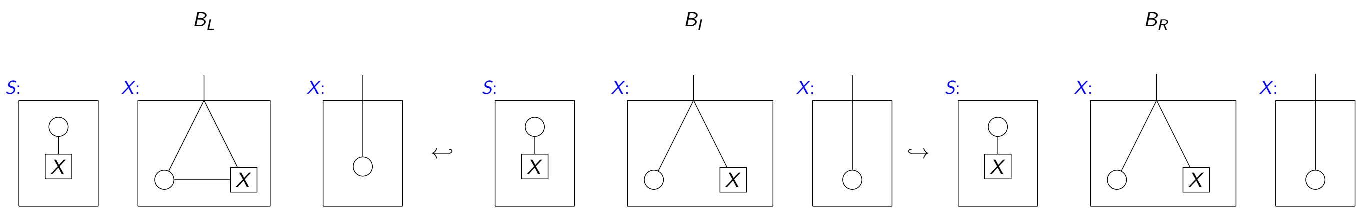


Step one

Encode equational schema as a grammar rewrite pattern.
This:



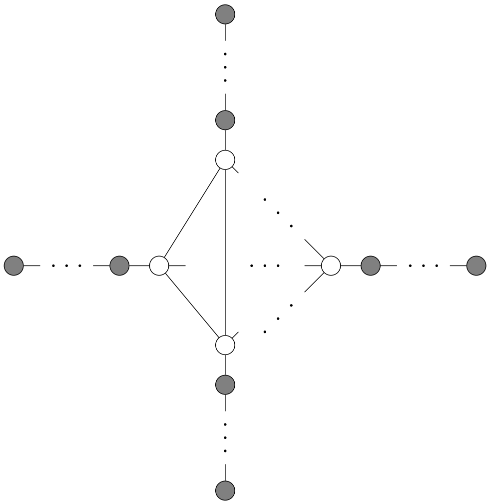
becomes this:



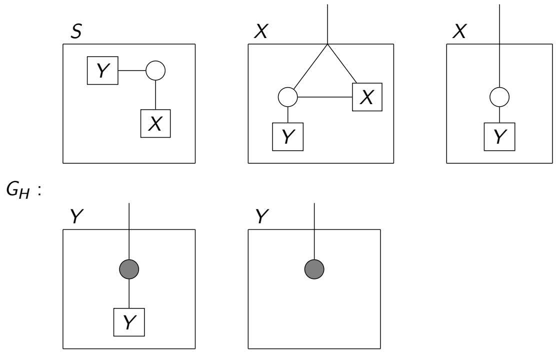
Step two

Encode the target family of string diagrams using a grammar

This:



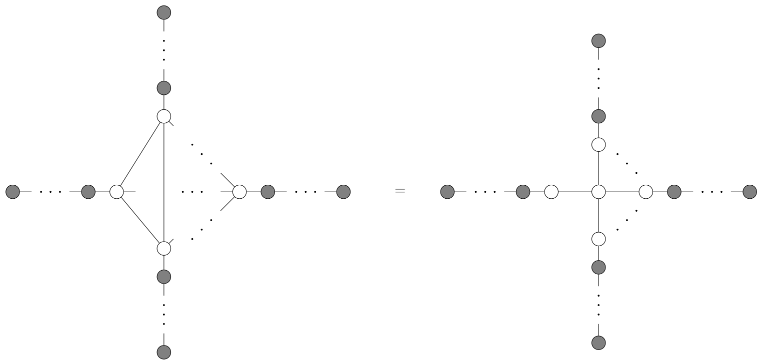
becomes this:



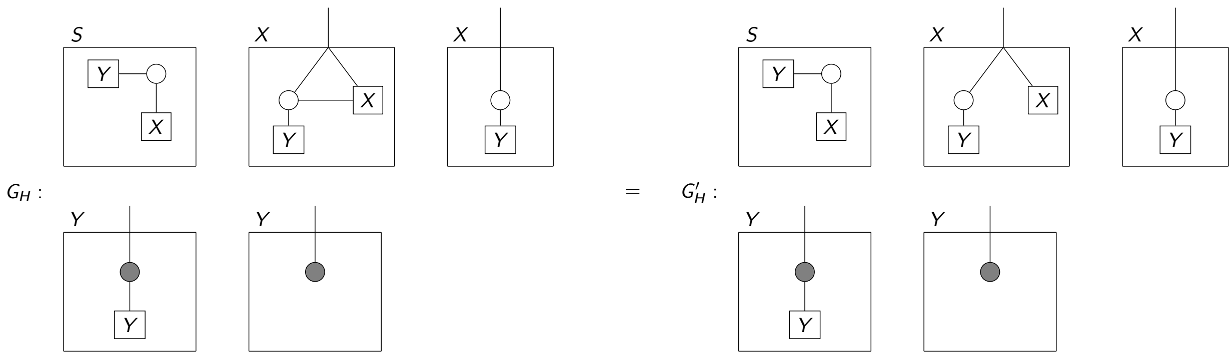
Step three

- Match the grammar rewrite rule into the target grammar and perform DPO rewrite (in **GGram**)
- Note, both the rewrite rules and the matchings are more restricted than what is required by adhesivity in order to ensure admissibility

This:



is then given by:



Admissibility

- Grammar rewriting as defined is admissible in the sense that the transformation of grammars respects their instantiations
- More formally:
- If a grammar G rewrites into a grammar G' via a grammar rewrite rule B, then:
  - Every concrete instantiation of B is a standard DPO rewrite rule on graphs
  - The language of B, denoted $L(B)$ is the set of all such DPO rewrite rules
  - The pair (G, G') forms a grammar pattern
  - For any concrete instantion H of G, a parallel concrete derivation H' exists for G'.
  - Then, the graph H' can be obtained from the graph H by applying some number of DPO rewrite rules on graphs from $L(B)$ in any order

Conclusion and Future Work

- Basis for formalized equational reasoning for context-free families of string diagrams.
  - Framework can handle equational schemas and it can apply them to equationally reason about families of string diagrams
- Identify more general conditions for grammar rewriting such that the desired theorems and decidability properties hold
- Implementation in software (e.g. Quantomatic proof assistant)
- Once implemented, software tools can be used for automated reasoing for quantum computation, petri nets, etc.

Thank you for your attention!