Higher-order rewriting of String Diagrams

Vladimir Zamdzhiev

21 April 2016
Families of string diagrams

- String diagrams 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

\[ \cdots \quad = \quad \cdots \]

- 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:

\[
\begin{array}{c}
\vdots \\
\vdots \\
\vdots \\
\end{array}
\quad = 
\begin{array}{c}
\vdots \\
\vdots \\
\vdots \\
\end{array}
\]

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

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

\[ \cdots \ = \ \cdots \]

\[ \Downarrow \]

\[ = \]

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

Example

\[
\begin{array}{cccc}
S &: X & \leftarrow & S &: X \\
& & & \\
B_L &: & & B_I \\
& & & & & & & & \\
B_R \\
& & & & & & & & \\
\end{array}
\]
Grammar rewrite pattern

Example

- Instantiation:
  
  S

  S

  S
Grammar rewrite pattern

Example

- Instantiation:

\[
S \quad \Rightarrow B_L \\
S \quad \Rightarrow B_I \\
S \quad \Rightarrow B_R
\]
Grammar rewrite pattern

Example

- Instantiation:

\[
S \quad \Rightarrow B_L
\]

\[
S \quad \Rightarrow B_R
\]

\[
S \quad \Rightarrow B_L
\]

\[
S \quad \Rightarrow B_R
\]
Grammar rewrite pattern

Example

- Instantiation:

\[
\begin{align*}
S & \Rightarrow B_L \\
& \Rightarrow B_L \\
& \Rightarrow B_L \\
S & \Rightarrow B_L \\
& \Rightarrow B_R \\
& \Rightarrow B_R \\
& \Rightarrow B_R \\
S & \Rightarrow B_R \\
& \Rightarrow B_R \\
& \Rightarrow B_R \\
& \Rightarrow B_R
\end{align*}
\]
Obtaining new equalities

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

\[
\begin{array}{c}
\cdots \\
\downarrow
\end{array}
= \begin{array}{c}
\vdots \\
\vdots
\end{array}
\]

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

Given an equational schema:

\[
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
= 
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\]

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

\[
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
= 
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\begin{array}{c}
\text{\ldots} \\
\text{\ldots} \\
\text{\ldots} \\
\end{array}
\]

Step one

Encode equational schema as a grammar rewrite pattern.
This:

\[
\begin{array}{c}
\xrightarrow{\ldots} \\
\xrightarrow{\ldots} \\
\xrightarrow{\ldots}
\end{array}
\]

becomes this:

\[
\begin{array}{c}
B_L \\
B_f \\
B_R
\end{array}
\]
Step two

Encode the target family of string diagrams using a grammar

This:

becomes this:

\[ S \]
\[ X \]
\[ X \]
\[ X \]
\[ G \]
\[ H \]
\[ Y \]
\[ Y \]
\[ Y \]
\[ Y \]
\[ V \]

G\(_H\) :

\[ Y \]
\[ X \]
\[ Y \]
\[ Y \]

Vladimir Zamdzhiev
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:

\[
S \xrightarrow{X} X \xrightarrow{X} X \xrightarrow{G} H =
\]

is then given by:

\[
G_H:
\]

\[
S \xrightarrow{X} X \xrightarrow{X} X \xrightarrow{Y} Y \xrightarrow{Y} Y \xrightarrow{Y} Y
\]

\[
G'_H:
\]

\[
S \xrightarrow{X} X \xrightarrow{X} X \xrightarrow{Y} Y \xrightarrow{Y} Y \xrightarrow{Y} Y
\]
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 reasoning for quantum computation, petri nets, etc.
Thank you for your attention!