Bridging computation and topology with diagram rewriting

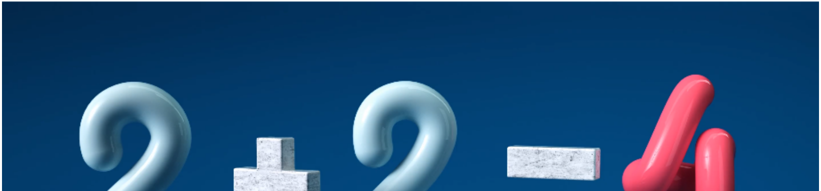
Amar Hadzihasanovic
Laboratory for Compositional Systems and Methods
Tallinn University of Technology

"ICT Means Business" Conference
2021

FOUNDATIONS OF MATHEMATICS

With Category Theory, Mathematics Escapes From Equality

Two monumental works have led many mathematicians to avoid the equal sign. Their goal: Rebuild the foundations of the discipline upon the looser relationship of "equivalence." The process has not always gone smoothly.



One of the biggest trends in contemporary mathematics

(via **higher category theory**):

One of the biggest trends in contemporary mathematics

(via **higher category theory**):

Replace equality with homotopy

(a **topological** notion of equivalence)

One of the biggest trends in contemporary mathematics

(via **higher category theory**):

Replace equality with homotopy

(a **topological** notion of equivalence)

Replace sets with higher-dimensional spaces

as the **native** objects of mathematics

A new field at the crossroads of mathematics and CS:

higher-dimensional rewriting

A new field at the crossroads of mathematics and CS:

higher-dimensional rewriting

Unifying rewriting, a fundamental mechanism of computation,

with homotopy, central to modern mathematics

The basic framework of rewriting:

1. we have some rewrite rules of the form $L \Rightarrow R$;

The basic framework of rewriting:

1. we have some rewrite rules of the form $L \Rightarrow R$;
2. we have a notion of matching L in some context $\mathcal{C}[L]$;

The basic framework of rewriting:

1. we have some rewrite rules of the form $L \Rightarrow R$;
2. we have a notion of matching L in some context $\mathcal{C}[L]$;
3. if we have a rewrite rule $L \Rightarrow R$, we can replace L with R in a context $\mathcal{C}[L]$, to obtain $\mathcal{C}[R]$

The basic framework of rewriting:

1. we have some rewrite rules of the form $L \Rightarrow R$;
2. we have a notion of matching L in some context $\mathcal{C}[L]$;
3. if we have a rewrite rule $L \Rightarrow R$, we can replace L with R in a context $\mathcal{C}[L]$, to obtain $\mathcal{C}[R]$

"Kindergarten" example: the rules

$$2 + 2 \Rightarrow 4, \qquad 4 + 0 \Rightarrow 4$$

can be used in the sequence of rewrites

$$2 + 2 + 0 \Rightarrow 4 + 0 \Rightarrow 4$$

The insight of higher-dimensional rewriting:

various kinds of rewriting can be unified as

diagram rewriting in different dimensions, The insight of higher-dimensional rewriting:

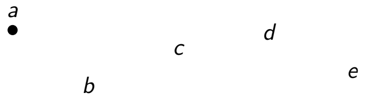
various kinds of rewriting can be unified as

diagram rewriting in different dimensions,

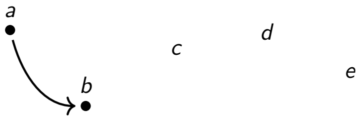
where the notion of diagram comes from higher category theory

and is compatible with a “homotopy” interpretation.

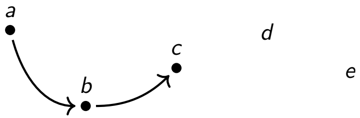
- Dimension 0: abstract rewriting



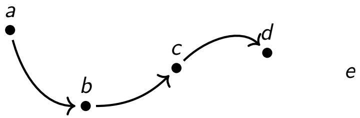
- Dimension 0: abstract rewriting



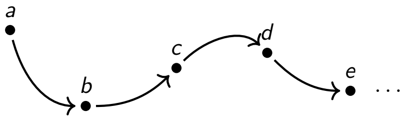
- Dimension 0: abstract rewriting



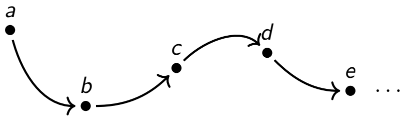
- Dimension 0: abstract rewriting



- Dimension 0: abstract rewriting

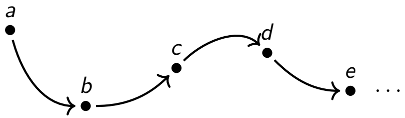


- Dimension 0: abstract rewriting

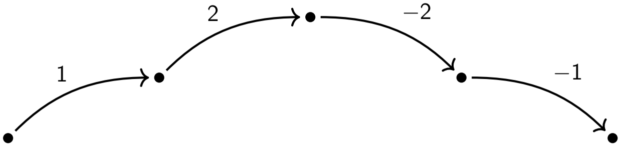


- Dimension 1: string rewriting

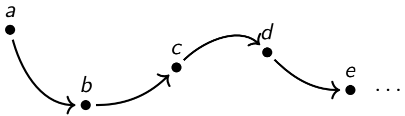
- Dimension 0: abstract rewriting



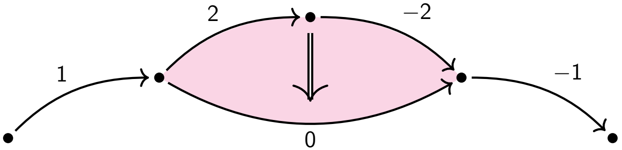
- Dimension 1: string rewriting



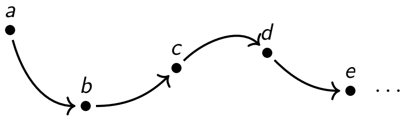
- Dimension 0: abstract rewriting



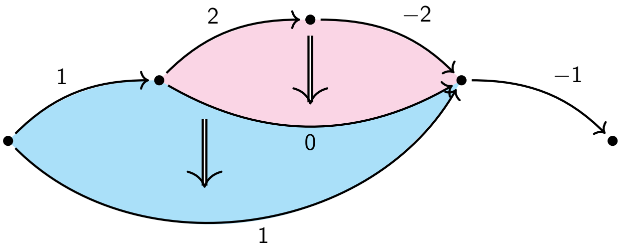
- Dimension 1: string rewriting



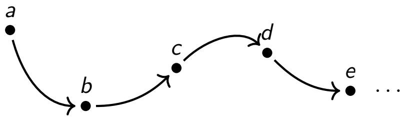
- Dimension 0: abstract rewriting



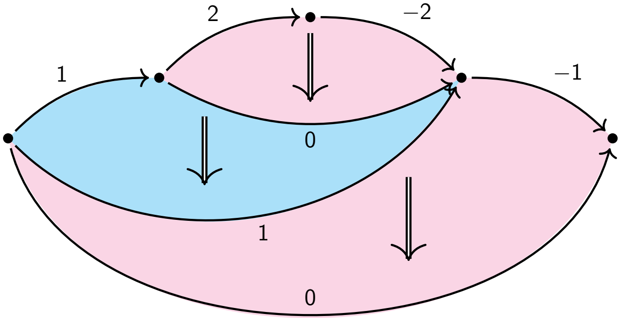
- Dimension 1: string rewriting



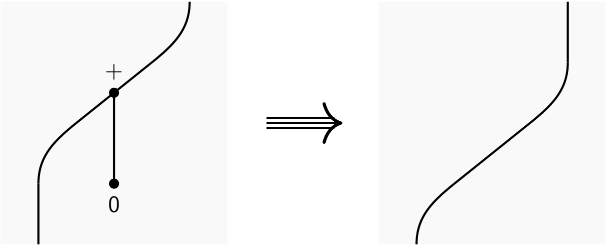
- Dimension 0: abstract rewriting



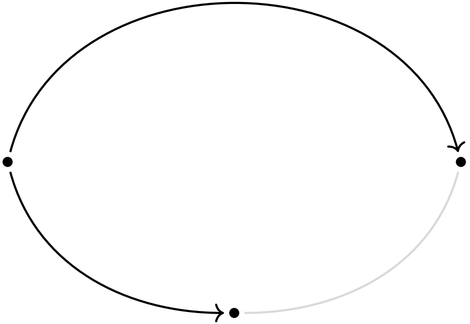
- Dimension 1: string rewriting

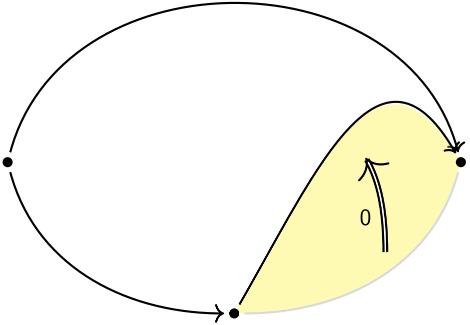


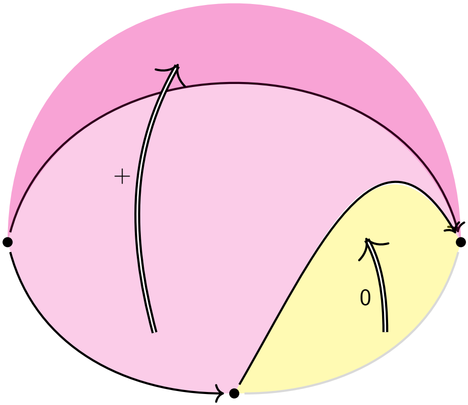
- Dimension 2: term rewriting $\subseteq$ string diagram rewriting

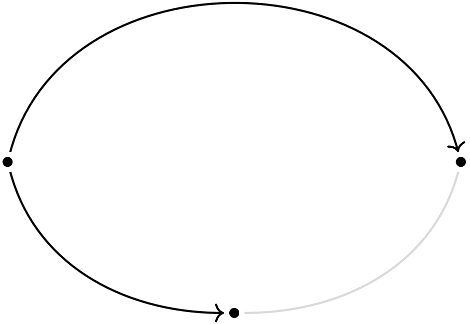


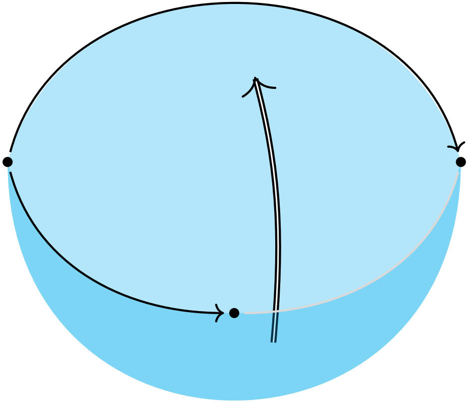
$$x + 0 \Rightarrow x$$

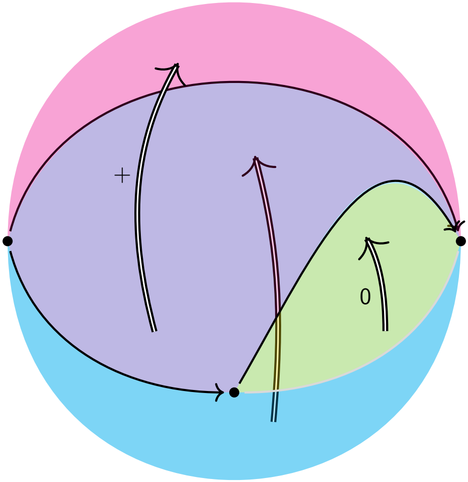


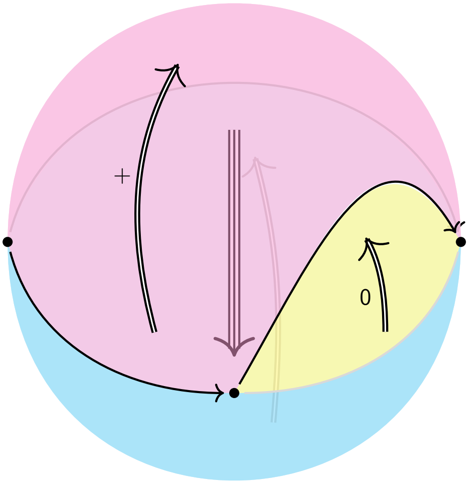

0


+
0






+
0


+
0

Theoretical work:

- build a flexible, rigorous toolkit for translating between rewriting and homotopy;
- use it to bring methods and intuition from topology to the study of computer programs

Theoretical work:

- build a flexible, rigorous toolkit for translating between rewriting and homotopy;
- use it to bring methods and intuition from topology to the study of computer programs

Towards a language for fundamental computer science that's closely integrated with 21st century mathematics

LICS: biggest conference in formal methods for computer science

2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)

The Smash Product of Monoidal Theories

Year: 2021, Volume: 1, Pages: 1-13
DOI Bookmark: 10.1109/LICS52264.2021.9470575

Authors

Amar Hadzihasanovic, Tallinn University of Technology,Department of Software Science,Tallinn,Estonia

Abstract

The tensor product of props was defined by Hackney and Robertson as an extension of the Boardman-Vogt product of operads to more general monoidal theories. Theories that factor as tensor products include the theory of commutative monoids and the theory of bialgebras. We give a topological interpretation (and vast generalisation) of this construction as a low-dimensional projection of a "smash product of pointed directed spaces". Here directed spaces are embodied by combinatorial structures called diagrammatic sets, while Gray products replace cartesian products. The correspondence is mediated by a web of adjunctions relating diagrammatic sets, pros, probs, props, and Gray-categories. The smash product applies to presentations of higher-dimensional theories and systematically produces higher-dimensional coherence data.

Applied work (with Diana-Maria Kessler):

- study of algorithmic aspects of higher diagram rewriting
- development of a new proof assistant for diagram rewriting

Goals

Our objective is to build a software toolkit for higher-dimensional diagram rewriting, with the main part being a proof assistant.

Goals

Our objective is to build a software toolkit for higher-dimensional diagram rewriting, with the main part being a proof assistant.

Our goals for the user are:

Goals

Our objective is to build a software toolkit for higher-dimensional diagram rewriting, with the main part being a proof assistant.

Our goals for the user are:

- Present a higher rewrite system.

Goals

Our objective is to build a software toolkit for higher-dimensional diagram rewriting, with the main part being a proof assistant.

Our goals for the user are:

- Present a higher rewrite system.
- Create "indexed" diagrams in the signature and have the software match rewrite rules as higher diagrams.

Goals

Our objective is to build a software toolkit for higher-dimensional diagram rewriting, with the main part being a proof assistant.

Our goals for the user are:

- Present a higher rewrite system.
- Create "indexed" diagrams in the signature and have the software match rewrite rules as higher diagrams.
- Be able to compute topological invariants of the systems considered, thanks to the connection to topology.

Data structures

1. We use a linear algebraic representation to encode the shape and orientation of diagrams in matrices.

Data structures

1. We use a linear algebraic representation to encode the shape and orientation of diagrams in matrices.
2. Some computations reduce to matrix multiplication.

Code

```python
import numpy as np
from scipy import linalg


class Diagram(object):

    def __init__(self, list_of_matrices):
        self.list_of_matrices = list_of_matrices  # list of 2D (or 1D) numpy arrays.

    def print(self):
        print("The diagram is: ")
        for i in range(0, len(self.list_of_matrices)):
            print("D" + str(i) + " is: \n", self.list_of_matrices[i])


class SubDiagram(object):

    def __init__(self, list_of_matrices, list_of_vectors):
        self.list_of_matrices = list_of_matrices
        self.list_of_vectors = list_of_vectors

    def print(self):
        print("Printing the subdiagram:")
        print("The list of matrices is: ")
        for i in range(0, len(self.list_of_matrices)):
            print("D" + str(i) + " is: \n", self.list_of_matrices[i])
        print("The list of vectors is: ")
        for j in range(0, len(self.list_of_vectors)):
            print("v" + str(j) + " is: \n", self.list_of_vectors[j])
```

(a)

```python
def split_vector(boundaries):
    omega_plus = np.zeros(boundaries.shape[0])
    omega_minus = np.zeros(boundaries.shape[0])

    for i in range(0, boundaries.shape[0]):
        if boundaries[i] == 1:
            omega_plus[i] = 1
        elif boundaries[i] == -1:
            omega_minus[i] = 1
        elif boundaries[i] != 0:
            raise ValueError("Non 0, 1 or -1 value in the boundaries values.")

    return (omega_plus, omega_minus)


def get_boundary(subdiagram, k): #TODO: Catch error if k is too big.
    print("Get_boundary.")
    list_of_matrices = subdiagram.list_of_matrices
    list_of_vectors = subdiagram.list_of_vectors
    n = len(list_of_matrices) - 1

    if list_of_matrices[n].ndim == 1:
        boundaries = np.outer (list_of_matrices[n], (list_of_vectors[n+1])).reshape(list_of_matrices[n].shape[0])
    else:
        boundaries = list_of_matrices[n] @ list_of_vectors[n+1]
    print("Boundary is: ", boundaries)

    if boundaries.ndim != 1:
        raise ValueError("Boundaries vector is not 1-dim.")

    (omega_plus, omega_minus) = split_vector(boundaries)
```

(b)

Figure

Algorithms

1. Algorithm to solve the isomorphism problem for diagram shapes.
    - Diagram traversal algorithm.

Algorithms

1. Algorithm to solve the isomorphism problem for diagram shapes.
    - Diagram traversal algorithm.
2. Algorithm for generating diagrams - we have identified an inductive construction for diagrams.

Diagram Traversal Algorithm 2

Diana Kessler

September 2021

- *Input*: A regular molecule, U, together with its input and output boundary at every level.
- *Output*: A linear ordering of the elements of U.
- *Aux*:
  - A list, o, with the ordering,
  - A list q with elements covering elements in o, waiting to be ordered.

In this algorithm, q works very much like a stack. Every time we add an element, e, in o, we add the elements that are covering e but are not in o into q. Sometimes it can be the empty list / nothing.

Generate Diagram

Diagram Generation

Diana Kessler

October 2021

1 Generate Atom

Input: 2 diagrams, U and V.
Output: a new diagram $U \implies V$.
Aux:

Convention:

1. The first argument of the program (U in this case) is the origin of the new cell, while the second is the target. The user chooses which of the diagrams get in the input boundary by providing it as the first argument.