

ZSS: Zipper logic revisited

Marius Buliga
mbuliga.github.io

09.06.2021

Abstract

Can we compute with tangles and Reidemeister moves? I explain where this problem comes from, why it is different than the usual way of using tangles in computation and then I propose the Zip-Slip-Smash (ZSS) graph rewriting system, which is an improvement of zipper logic arXiv:1405.6095, and prove that it allows universal computation using tangles and zippers, via directed Interaction Combinators.

Introduction

This is based on the transcript of a talk from September 2020, which I gave at the [Quantum Topology Seminar](#) organized by Louis Kauffman.

There is a [video of the talk](#) and a [github repository](#) associated with ZSS.

The problem: is to compute with Reidemeister moves, i.e by use of a graph rewrite system which contains the tangle diagrams as graphs and Reidemeister moves as graph rewrites.

Can we do universal computation with this?

Where does this problem come from?

This is not the usual way to do computation with tangles. The usual way, Figure 1, is that we have a circuit which we represent as a tangle, a knot diagram, where the crossings and maybe caps and cups are operators which satisfy the algebraic equivalent of the R moves.

Each circuit represents a computation. When we have two circuits, then we can say that they represent equivalent computations when we can turn one into another by using R moves.

As an example, in graphical formalisms used for quantum computation [like ZX calculus](#) [10] we have preparation (cups), detectors (caps) and crossings are quantum gates. R moves and cup-cap annihilation graph rewrites transform a computation into another. The well known example of teleportation shows

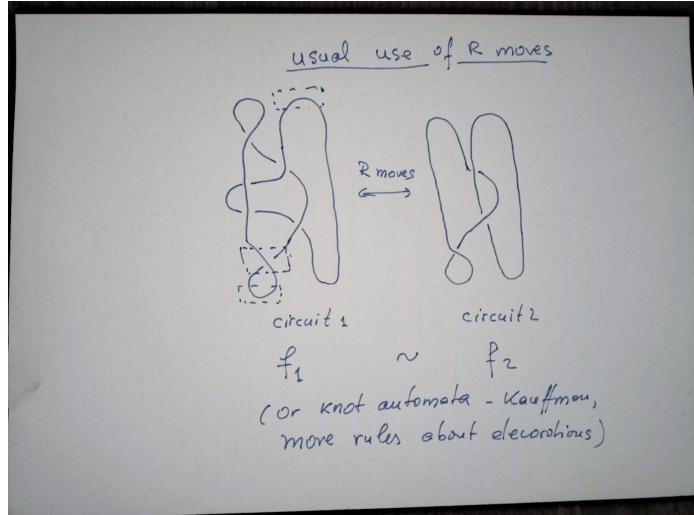


Figure 1: usual use of R moves

that two computation are equivalent. The first computation has associated a graph which represents the teleportation experiment. It is not clear what is the effect of this computation. The second computation is obviously clear to us (teleportation) but it is not clear how we can realize this computation in practice. The equivalence (not via R moves in this case, but involving the annihilation of a cup with a cap) between the two graphs show that these two graphs represent the same computation, therefore we obtain an explanation why the experimental setting achieves the effect of teleportation.

We see here that the graph rewrites don't compute by themselves, instead they are used to prove that two computations are equivalent. The graph rewrites do not compute!

My source of interest in the problem if we can compute with the R moves comes from emergent algebras, introduced in [3] [arXiv:0907.1520](https://arxiv.org/abs/0907.1520) and their graphical representations using tangles [4] [arXiv:1103.6007](https://arxiv.org/abs/1103.6007). An e.a. is a combination of algebraic and topological information, see Figure 2.

See also [6] [arXiv:1807.02058](https://arxiv.org/abs/1807.02058) for an associated lambda calculus style term rewriting system.

For convenience, I reproduce here the short description of emergent algebras from [5] [A problem in emergent algebras](#).

Definition. An emergent algebra over a set X is a family of idempotent left quasigroup operations over X , indexed by the group Γ , which satisfy the following algebraic and topological axioms (R1), (R2), (act) and (em).

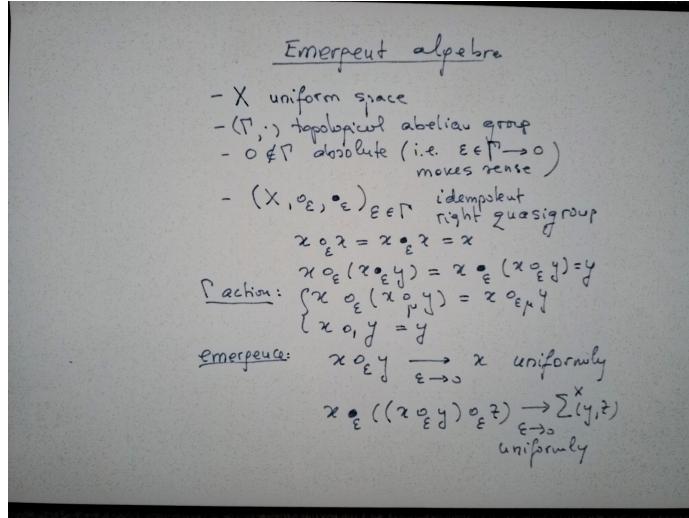


Figure 2: definition of emergent algebras

For $a \in \Gamma$ let's denote the left quasigroup operations indexed by a with \circ_a and \bullet_a . Therefore (X, \circ_a, \bullet_a) is an idempotent left quasigroup:

$$(R1) \quad x \circ_a x = x$$

$$(R2) \quad x \circ_a (x \bullet_a y) = x \bullet_a (x \circ_a y) = y$$

(act) algebraic axioms which relate the operations:

$$x \circ_a (x \circ_b y) = x \circ_{ab} y$$

$$x \circ_1 y = y$$

$$x \circ_{1/a} y = x \bullet_a y$$

On X we define the following operations which we will use later:

- the approximate difference

$$\Delta_a^x(y, z) = (x \circ_a y) \bullet_a (x \circ_a z)$$

- the approximate sum

$$\Sigma_a^x(y, z) = x \bullet_a ((x \circ_a y) \circ_a z)$$

- the approximate inverse

$$inv_a^x y = (x \circ_a y) \bullet_a x$$

Finally we have the topological axioms.

(em) X is an uniform space; as a converges to 0

$(x, y) \mapsto x \circ_a y$ converges uniformly to $(x, y) \mapsto x$

$(x, y, z) \mapsto \Delta_a^x(y, z)$ converges uniformly to a function $(x, y, z) \mapsto \Delta^x(y, z)$

$(x, y, z) \mapsto \Sigma_a^x(y, z)$ converges uniformly to a function $(x, y, z) \mapsto \Sigma^x(y, z)$

By the usual correspondence between quandles and crossings, in emergent algebras we can perform the Reidemeister 1 and 2 moves, but not the R3. On the other hand, we can pass to the limit with the parameter which decorates the operations (or crossings).

This passage to the limit produces emergent objects and properties. Here is how.

In Figure 3 is the configuration which gives the approximate sum.

The crossings represent emergent algebra operations. The graphical representation tells that as epsilon goes to 0 you obtain in the limit some gate (the sum gate, drawn as a hexagon) which has 3 inputs and 3 outputs.

We say that the sum is an emergent object because it emerges from the limit. We can also define in the same way the derivative of a function, graphically, as in Figure 4.

We can define not only emergent objects, but also emergent properties.

As an example, in Figure 5 we use the R2 rewrites and passage to the limit to prove that the addition is associative.

The mechanism is: you pass to the limit from left to right of the figure and you use R2 rewrites from up to down, to obtain the associativity of the sum operation in graphical form.

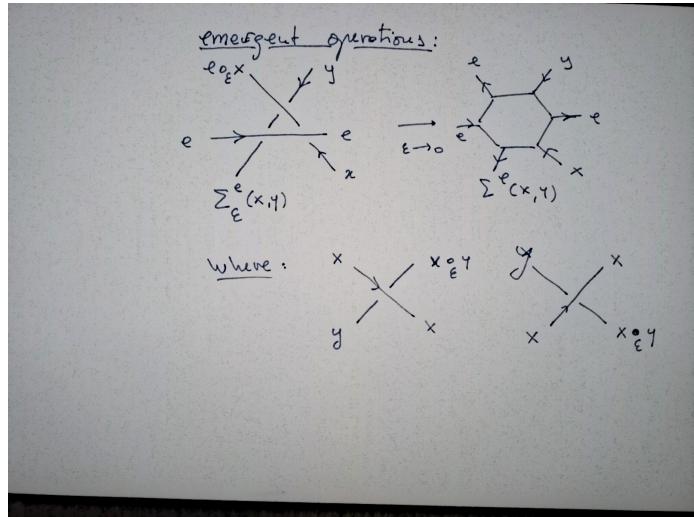


Figure 3: emergent sum

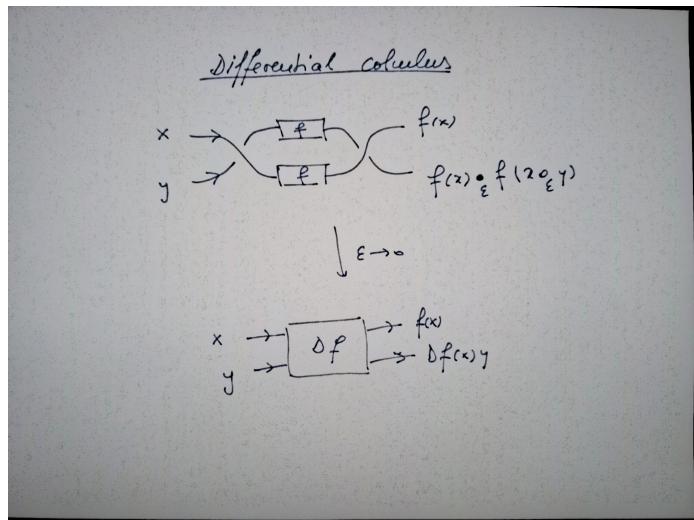


Figure 4: definition of the derivative

Another example in Figure 6: if you define a new crossing (relative crossing) then you can pass to the limit and you can prove that, based on e.a. axioms, including a passage to the limit, the R3 emerges.

Indeed, what we see is that the relative crossing satisfies R1, R2 and R3, compared with the original crossing which satisfies only R1 and R2.

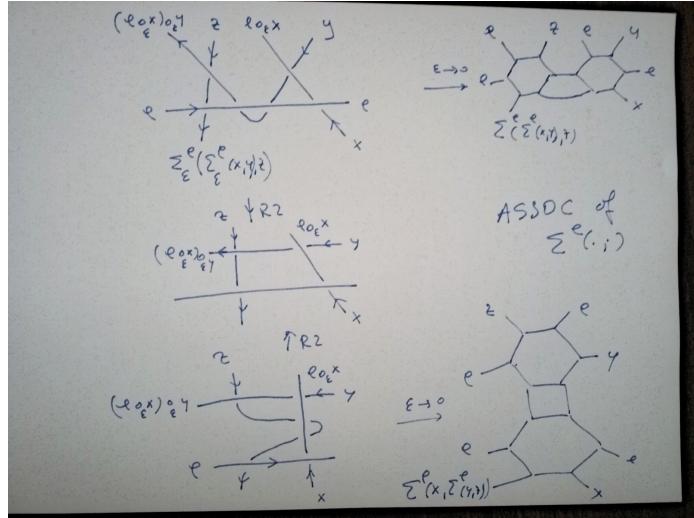


Figure 5: associativity as emergent property

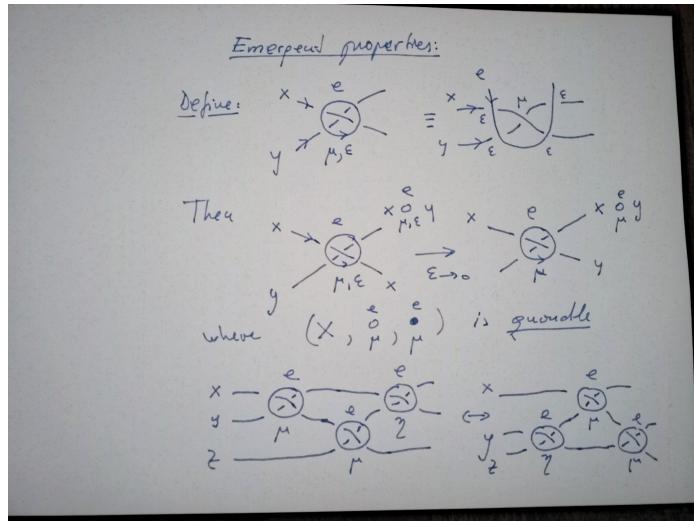


Figure 6: R3 emerges from R1 and R2 and a passage to the limit

With e.a. we can do differential calculus. We use only R1, R2 and a passage to the limit. It is a differential calculus which is not commutative.

There are interesting classes of emergent algebras, Figure 7:

- linear e.a. correspond to calculus on conical groups (Carnot groups are a particular class encountered in sub-riemannian geometry, for example)

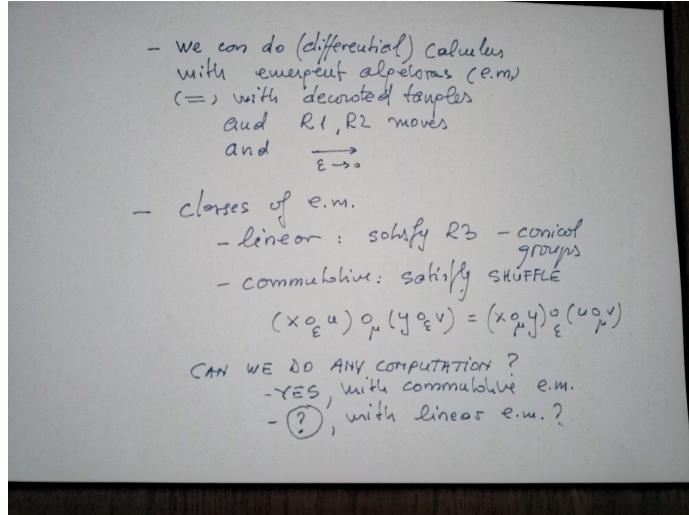


Figure 7: Classes of emergent algebras

- commutative e.a. which satisfy SHUFFLE (leads to the usual calculus in topological vector spaces) In this class you can do any computation ([Pure See](#))

Technically, what I want to know is: can you do universal computation in the linear case? This corresponds to the initial problem.

What means universal computation?

There are many, but among them, 3 ways to define what computation means. They are equivalent in a precise sense.

Lambda calculus is a term rewrite system.

A Turing machine is an automaton.

Lafont' [Interaction Combinators](#) [11] is a graph rewrite system, where you use graphs with two types of 3 valent nodes and one type of 1 valent node. Mind that we work with [port graphs](#) [1], where nodes have numbered ports.

Not shown in Figure 9 are IC rewrites which delete nodes.

Lafont proves that his graph rewrite system is universal because he can implement Turing machines.

There is a lot of work to implement lambda calculus in a graph rewrite system, in particular in interaction combinators. The reality is that this is extremely difficult, in the sense that there are solutions, but the solutions are not what we want. You can transform a lambda term into a graph and then reduce it with

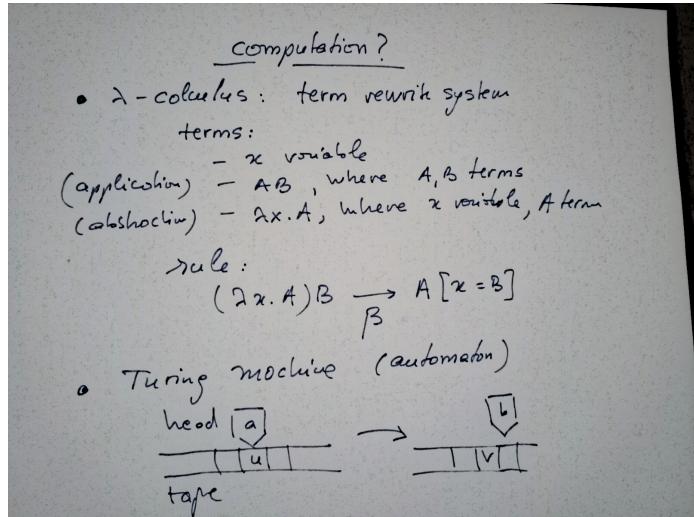


Figure 8: Lambda calculus, Turing machines

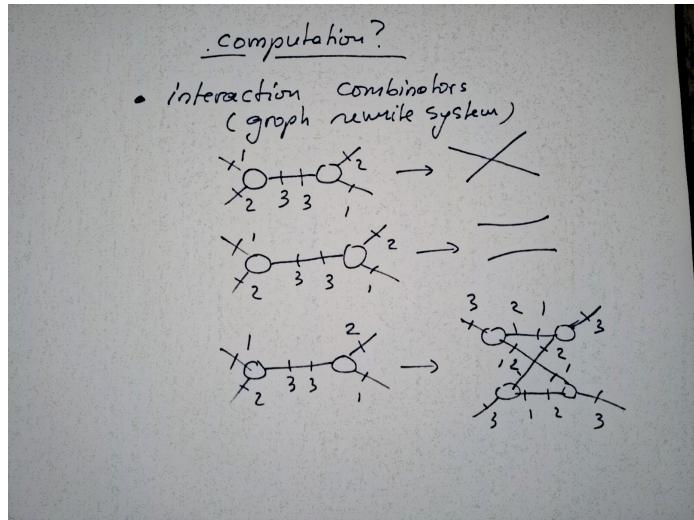


Figure 9: Lafont' Interaction Combinators

the graph rewrite system of Lafont, for example, and then you can decorate the edges of the graph so that you can retrieve the result of the computation. But while the graph rewrite system and the algorithm of rewrite applications are local, the parsing from lambda calculus to graphs and back are non local (there is no a priori upper bound of the number of nodes and edges involved). Differently from the graph rewrite systems we are interested in, term rewrite

systems are non-local. See [Asemantic computing](#).

We have 3 definition of what computation means, by 3 different models, which are equivalent only if you add supplementary hypotheses. For me IC is the most important one, but we don't know yet how to compute with IC only.

Let me reformulate the problem of if we can compute with R moves.

We can associate a knot quandle to a knot diagram, Figure 10, simply by naming the arcs, then we get a presentation of a quandle.

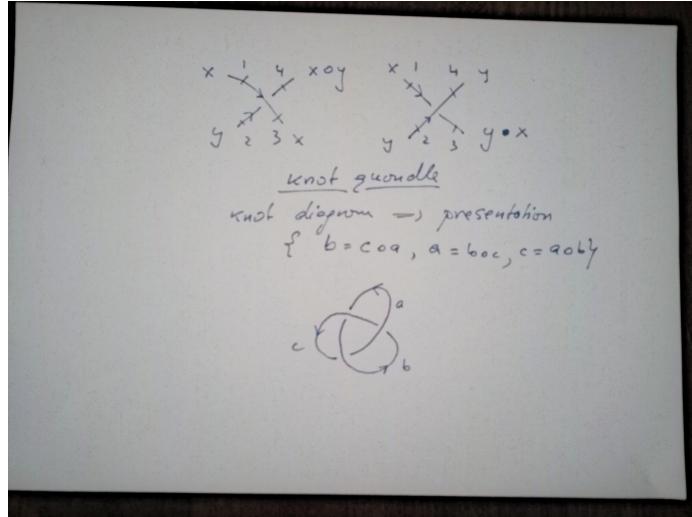


Figure 10: Knot quandle

The presentation of a quandle, Figure 11 and Figure 12 is invariant with respect to the permutation of relations or the renaming of the arcs. There is a small problem, we have to introduce fanin and fanout nodes too. For example when an arc passes over two crossings, we have a hidden fanout. The solution is to use a different notation and FIN (fanin) and FO (fanout) nodes. This turns the presentation into a [mol file \[7\]](#), like described in the following figure.

Can we compute with that?

Theorem. If there is a parser from lambda calculus to knot diagrams, such that any lambda calculus rewrite is parsed to a pair of knot diagrams which are equivalent under the Reidemeister moves, and such that there is a term sent to a diagram of the unknot, then all lambda terms are sent to diagrams of the unknot.

The proof is simple. Let A be the non empty set of all lambda terms which are parsed to a diagram of an unknot and let B the set of all other lambda terms. By Haken we have an algorithm to detect the unknot and the Scott-Curry theorem tells us that B has to be empty.

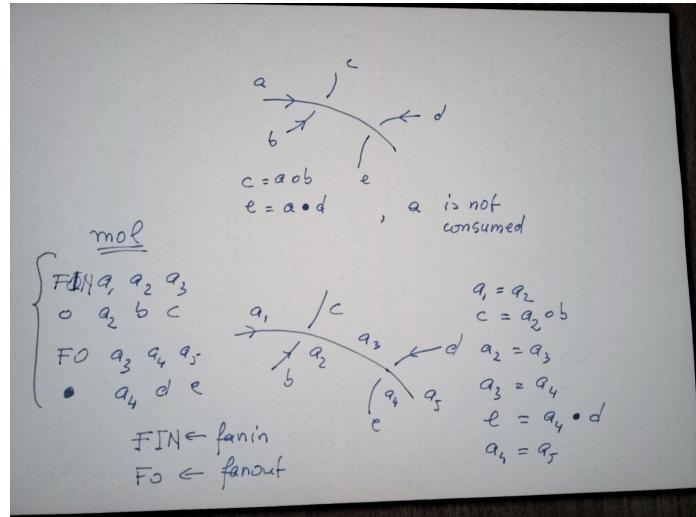


Figure 11: Quandle presentation in mol notation I

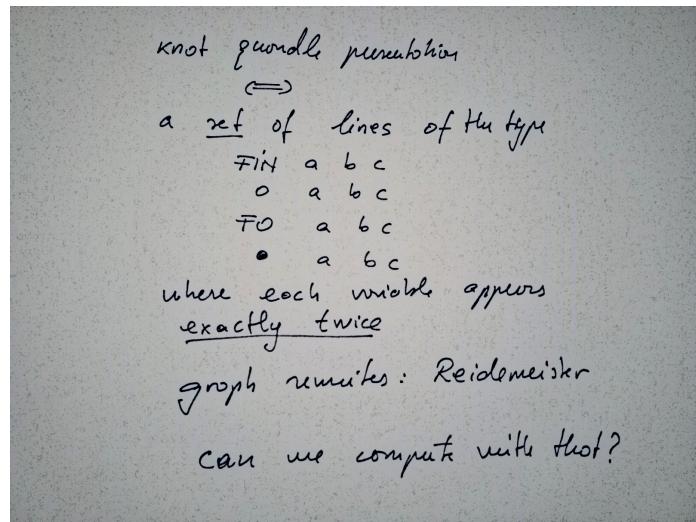


Figure 12: Quandle presentation in mol notation II

This shows that we can't hope to do computations with tangle diagrams and R moves only, with the algorithm of random application of rewrites (or with any local algorithm), because if we could then we can also turn any graph representing a lambda term to the unknot.

We can compute with knot diagrams, but in a stupid way: if we use diagrams as a notational device.

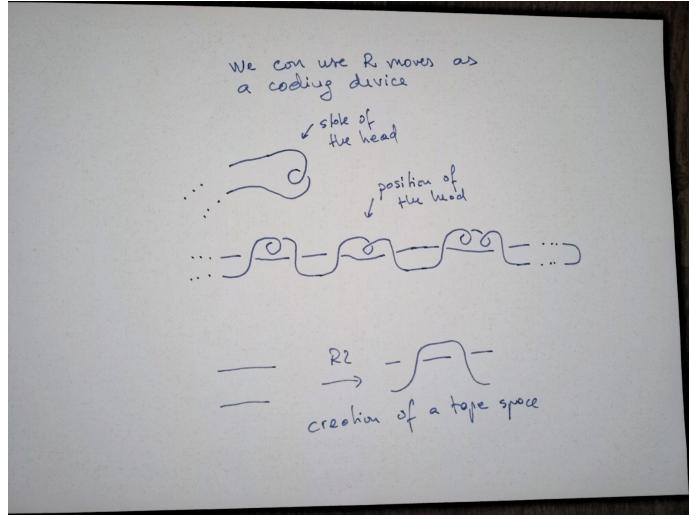


Figure 13: Turing machine with tangles and R moves

Indeed, as described in Figure 13, take an unknot, flatten it and use parts of it for the head state and other parts for the tape symbols. The computation is then realized via R moves, but there are sequences of R moves which turn the diagram into the unknot, which do not correspond to a Turing machine computation. You may add variations where we use such diagrams for headless Turing machines, or even for another universal computation model, the SKI combinator calculus [via chemSKI](#). We can't hope to use only the R moves, in a local way, to do universal computation.

Zip-Slip-Smash

I argue that we need to introduce a way to reconnect edges. For this I introduce zippers. The idea is that a zipper is a thing which has two non-identical parts, so it's made of two half-zippers, which are not identical. There will be two types of zippers: black and white. We also have ends (black and white) which are 1 valent, and of course crossings of two types, which are 4 valent nodes. In Figure 14 we see the elements of ZSS.

There are 4 types of half-zippers (two white, two black). In Figure 15 you see them also in the mol notation.

The L and A half-zippers are related to the lambda calculus abstraction and application operations, like in [8] [chemlambda](#).

We also have crossings, which are 4 valent nodes, Figure 16. Each crossing is represented in the mol notation by a pair of lines.

This corresponds to the usual way to see a crossing (which is 4 valent) as if it is

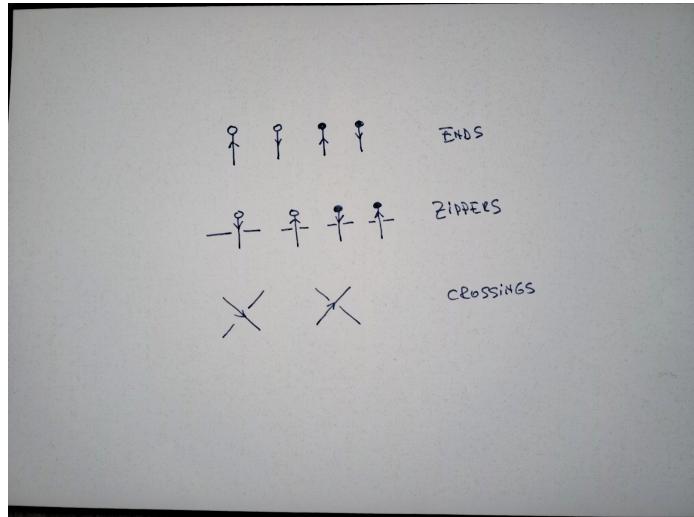


Figure 14: The elements of ZSS

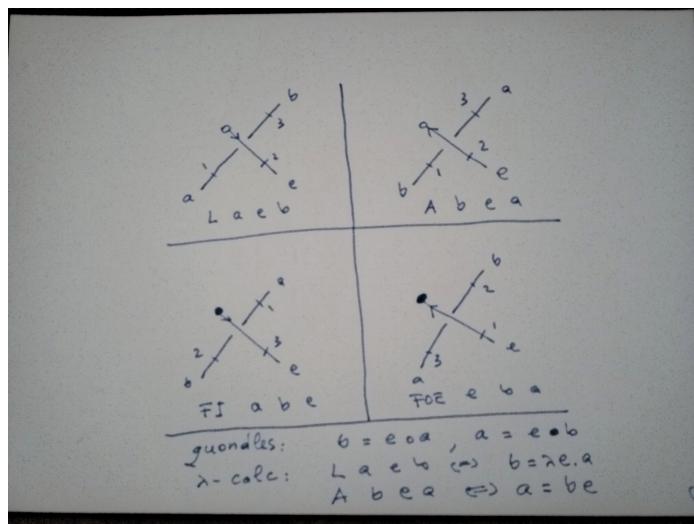


Figure 15: 4 types of half-zippers

a pair of two 3 valent nodes, one of them being a fanin (FIN) or a fanout (FO).

Now let's see the rewrites which give the name of ZSS: zip, slip, smash.

There are two **zip rewrites**. The first one, Figure 17, uses only white half-zippers.

It can be used to make diagrams and rewrites which really look like zippers,

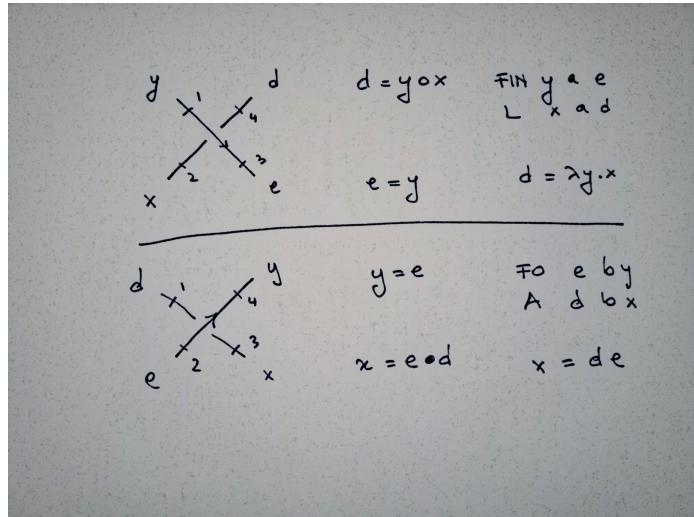


Figure 16: 2 types of crossings

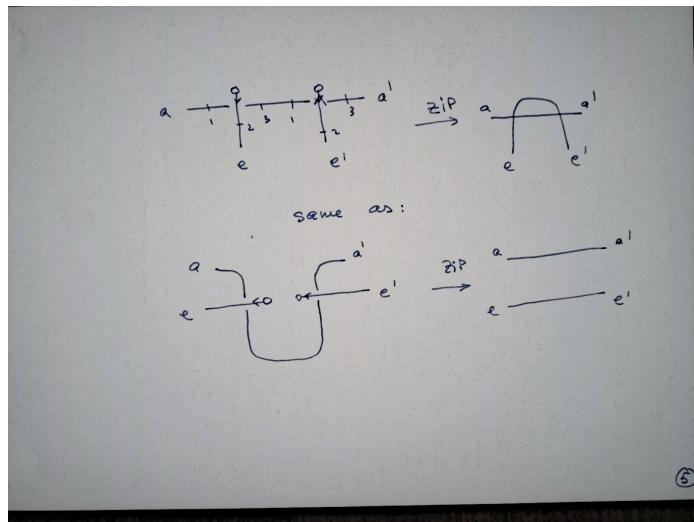


Figure 17: First zip rewrite

Figure 18.

The second zip rewrite, Figure 19, uses only black half-zippers:

With this we can make black zippers as well.

The **slip rewrites** are introduced, Figure 20, because there is a graphical ambiguity in the drawing of a half-zipper. We see there an end (black or white)

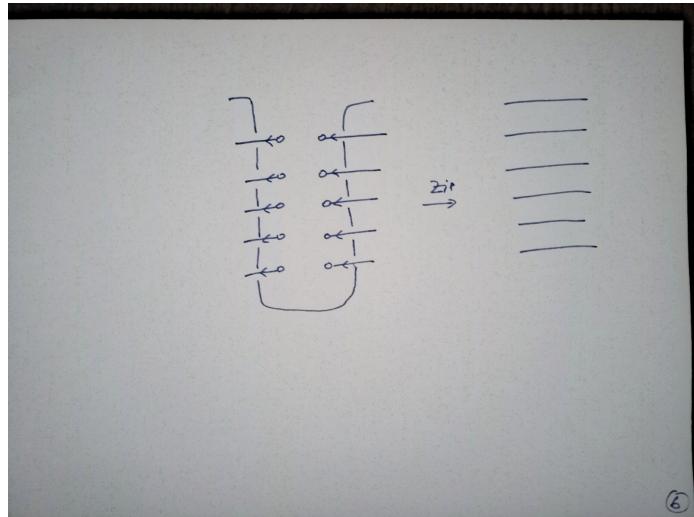


Figure 18: White zipper

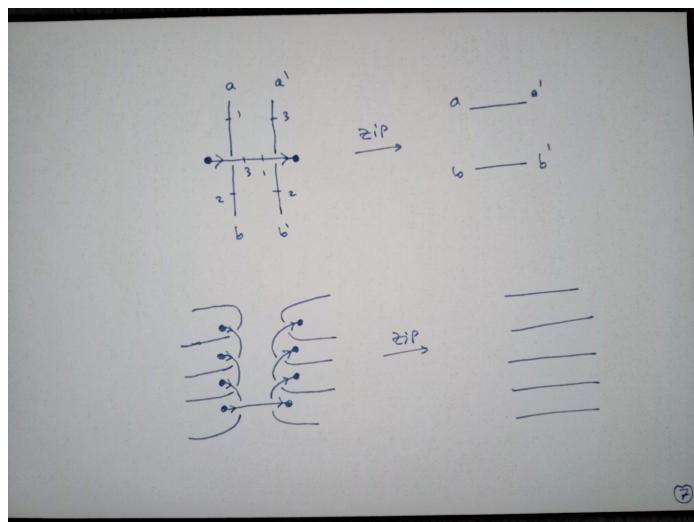


Figure 19: Second zip rewrite and black zippers

and a crossing, or is there a black or white half-zipper?

The slip rewrites tell us that ends (of any color) which pass over the crossing are the same as half-zippers, and also that ends which pass under the crossing ... slip under and the crossing is destroyed. S. Carter mentions that such rewrites appear also in [9] Geometric Interpretations of Quandle Homology arXiv:math/0006115v1 .

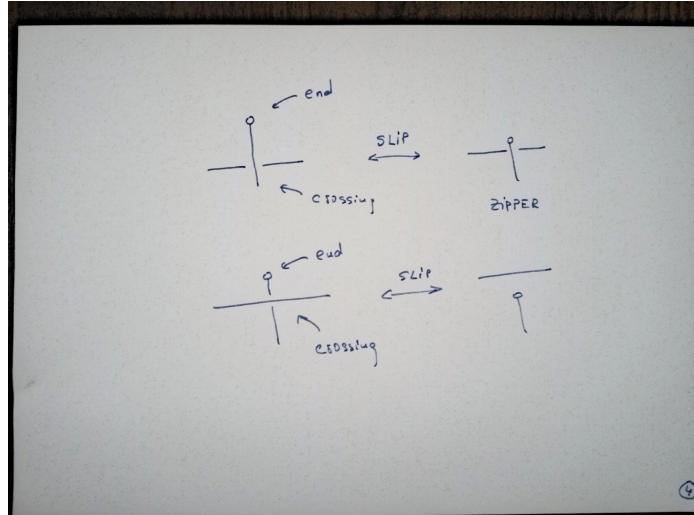


Figure 20: The slip rewrites

The **smash rewrites** create half-zippers. Imagine that you smash a crossing with a hammer and it breaks down into two half-zippers. (Mind that all rewrites are bi-directional though, so you can also turn a pair of half-zippers into a crossing). The first smash rewrite is in Figure 21.

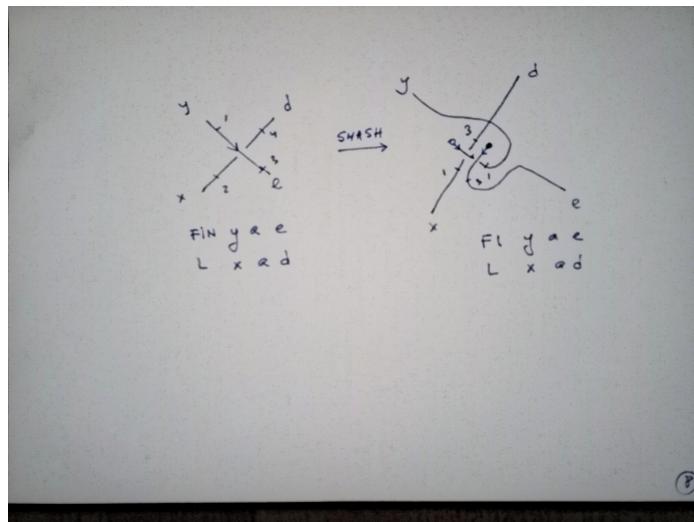


Figure 21: The first smash rewrite

We see that, in the mol notation, it turns a FIN fanin into a FI 3 valent node, in the presence of a L node. Conversely, a pair of half-zippers (black and white)

connected as shown, turns into a crossing.

The second smash rewrite, Figure 22, involves the other pair of black and white half-zippers.

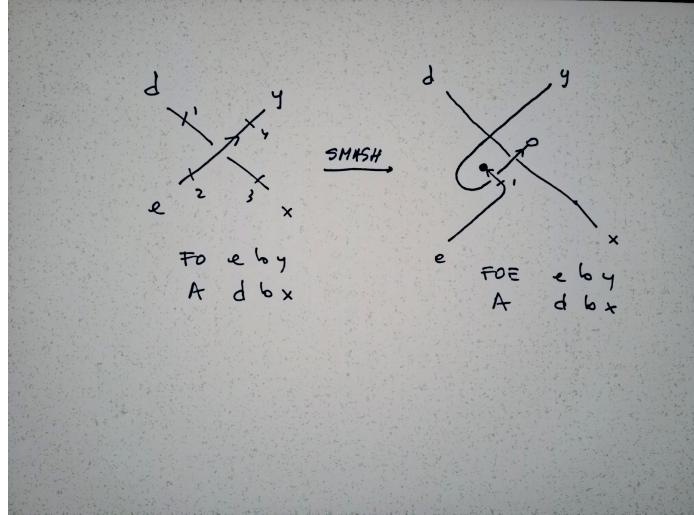


Figure 22: The second smash rewrite

Finally, we have the **crossings rewrites!** They are the R1, R2, R3 familiar graph rewrites which involve only crossings.

In conclusion ZSS is a graph rewrite system which enlarges the Reidemeister rewrites with supplementary ones (zip, slip, smash) which allow reconnection of edges, and tangle graphs with some 3 valent and 1 valent nodes.

ZSS is an improved version of [2] [Zipper logic \(figshare\)](#) or [arXiv:1405.6095 \[math.CO\]](#).

Indeed, the original zipper logic used tangles with zippers, but there were also fanins and fanout nodes. Here the fanins and fanouts are embedded into crossings. Also, the duplication rewrites of zipper logic are not local, in the sense that whole combinations of half-zippers duplicate at once. Here we have rewrite duplications which are built from more simple pieces.

As an example, in the Figure 23 is described how we can achieve a duplication rewrite (L-FOE) in ZSS:

We can give a similar proof of the duplication FI-A. Together with the two zip rewrites and with the slip rewrites, we obtain all rewrites of dirIC.

Theorem. ZSS is universal, because it implements directed [Interaction Combinators \(dirIC\)](#).

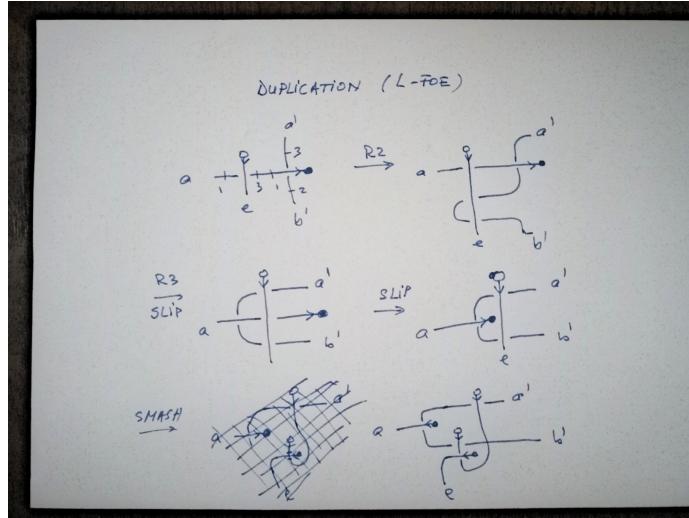


Figure 23: Duplication rewrite in ZSS (here L-FOE)

Recall that once we have the nodes L, FI, A, FOE and the mentioned rewrites, then indeed we can reconstruct Lafont' Interaction combinators by grouping A, L and FI, FOE, Figure 24.

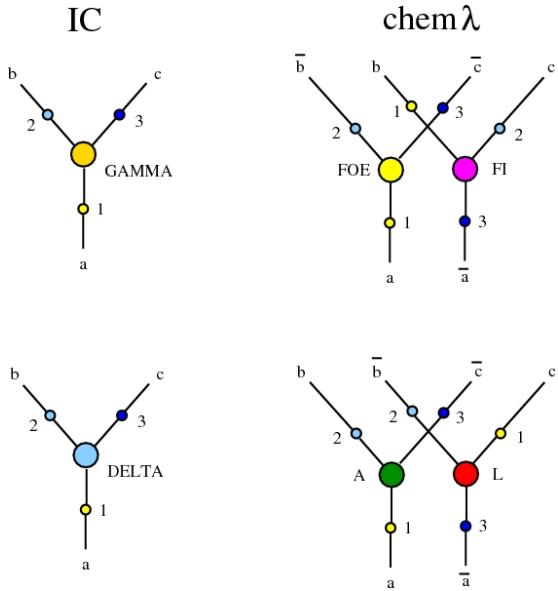


Figure 24: Interaction Combinators and dirIC

Not shown, the IC rewrites which delete nodes correspond to slip rewrites in

ZSS.

It is very interesting that in ZSS we obtain after smash rewrites pairs of L, FI or A,FOE nodes. Compare with the groupings L,A or FI,FOE of dirIC.

Moreover in ZSS we have the Reidemeister rewrites for crossings, which are really useful, as shown in the L-FOE rewrite, which is obtained from R2, R3, slip and smash rewrites.

Questions

- How can we add a passage to the limit like in emergent algebras?
- How is this compatible with Pure See?
- Is smash a rewrite which represents a measurement, if we were to try to use ZSS for quantum computing?

References

- [1] A. Bawden, Implementing Distributed Systems Using Linear Naming, [A.I. Technical Report No. 1627](#), MIT, (1993)
- [2] M. Buliga, Zipper logic, [doi:0.6084/m9.figshare.1032660.v1](#), [arXiv:1405.6095](#)
- [3] M. Buliga, Emergent algebras, [arXiv:0907.1520](#)
- [4] M. Buliga, Computing with space: a tangle formalism for chora and difference, [arXiv:1103.6007](#)
- [5] M. Buliga, [A problem in emergent algebras](#)
- [6] M. Buliga, The em-convex rewrite system, [arXiv:1807.02058](#)
- [7] M. Buliga, Artificial chemistry experiments with chemlambda, lambda calculus, interaction combinators, [arXiv:2003.14332](#)
- [8] M. Buliga, [Graph rewrites, from emergent algebras to chemlambda](#), [arXiv:2007.10288](#)
- [9] J.S. Carter, S. Kamada, M. Saito, Geometric Interpretations of Quandle Homology, [arXiv:math/0006115v1](#)
- [10] B. Coecke, R. Duncan, Interacting quantum observables: Categorical Algebra and Diagrammatics, [doi:10.1088/1367-2630/13/4/043016](#), [arXiv:0906.4725](#)
- [11] Y. Lafont, [Interaction Combinators](#), *Information and Computation* **137**, 1, 69-101, (1997)