

# Linear lambda terms as invariants of rooted trivalent maps

Noam Zeilberger

February 2, 2016

## Abstract

The main aim of the article is to give a simple and conceptual account for the correspondence between ( $\alpha$ -equivalence classes of) closed linear lambda terms and (isomorphism classes of) rooted trivalent maps on compact oriented surfaces without boundary, as an instance of a more general correspondence between linear lambda terms with a context of free variables and rooted trivalent maps with a boundary of free edges. We begin by recalling a familiar diagrammatic representation for linear lambda terms, and explain how these diagrams may be interpreted formally as 1-cells in a symmetric monoidal closed bicategory equipped with a reflexive object. From there, the “easy” direction of the correspondence is a simple forgetful operation which erases annotations on the diagram of a linear lambda term to produce a rooted trivalent map. The other direction views linear lambda terms as *invariants* of their underlying rooted trivalent maps, reconstructing the missing information through a Tutte-style topological recurrence on maps with free edges. As an application, we show how to use this analysis to enumerate *bridgeless* rooted trivalent maps as linear lambda terms *containing no closed subterms*, and conclude with a natural reformulation of the Four Color Theorem as a statement about lambda calculus.

## 1 Introduction

This paper follows recent work on the combinatorics of linear lambda terms, which has uncovered various connections to the combinatorics of rooted maps. Currently, it is known that there exist size-preserving correspondences between all of the following pairs of families of objects, some with explicit bijections but all at least at the level of generating functions:

Family of rooted maps	Family of lambda terms	OEIS [14]
rooted trivalent maps	linear lambda terms [3]	A062980
planar rooted maps	normal planar lambda terms [24]	A000168
rooted maps	normal linear lambda terms / $\sim$ [25]	A000698

Since the “meaning” of these connections is still not completely clear, however, my aim here is to revisit the basic situation of rooted trivalent maps and propose a slightly more conceptual account of the bijection originally given by Bodini, Gardy, and Jacquot [3], one which hopefully suggests some directions for further exploration. The main ideas I want to describe are:

1. The bijection between closed linear lambda terms and rooted trivalent maps (on compact oriented surfaces without boundary) is really an instance of a more general bijection that relates linear lambda terms with free variables to rooted trivalent maps with a marked boundary of free edges.
2. If we represent a linear lambda term as a *string diagram*, then the corresponding rooted trivalent map is obtained simply by forgetting some information encoded locally at vertices. Conversely, this extra information may be globally reconstructed through a decomposition of rooted trivalent maps with free edges by iterated examination of the root (similar in spirit to Tutte’s seminal analysis of rooted planar maps [21]). In effect, a linear lambda term can be seen as a topological *invariant* of a rooted trivalent map (analogous to, say, the Tutte polynomial of a graph), which is moreover complete in the sense that it characterizes the rooted trivalent map up to isomorphism.



A nice feature of combinatorial maps is that it is easy to compute their *genus*.

**Definition 2.2.** Let  $M$  be a combinatorial map. The **genus**  $g$  of  $M$  is defined by the Euler-Poincaré formula  $c(v) - c(e) + c(f) = 2 - 2g$ , where  $v$ ,  $e$ , and  $f$  are respectively the vertex, edge, and face permutations associated to  $M$ , and  $c(\pi)$  counts the number of cycles in the cycle decomposition of  $\pi$ . (For example, we have  $c(v) - c(e) + c(f) = 6 - 9 + 5 = 2$  for the genus  $g = 0$  map of Figure 1.)

In the rest of the paper we will be focused on *trivalent* maps (also called cubic maps), which can be defined by the algebraic condition that the vertex permutation is fixed point-free and of order three.

**Definition 2.3.** Let  $\mathcal{T}$  be the group  $\mathcal{T} \stackrel{\text{def}}{=} \langle v, e \mid v^3 = e^2 = 1 \rangle$ . A **trivalent map** is a transitive  $\mathcal{T}$ -set on which the generators  $v$  and  $e$  act without fixed points.

Moreover, we will always be speaking about *rooted* maps.

**Definition 2.4.** A **rooted** (trivalent) map is a (trivalent) map  $M$  equipped with a distinguished element  $r \in M$ . An isomorphism of rooted maps  $f : (M, r) \rightarrow (M', r')$  is an isomorphism of maps  $f : M \rightarrow M'$  which preserves the root  $f(r) = r'$ .

Topologically, a rooting of a map can be described as the choice of an edge, vertex, and face all mutually incident, or equivalently as the choice of an edge together with an orientation of that edge. The original motivation for the study of rooted maps in combinatorics was that they are *rigid* objects (i.e., they have no automorphisms other than the identity) and hence are easier to count (see [22, Ch. 10]), but it is also worth remarking that there is a general correspondence

$$\text{pointed transitive } G\text{-sets} \leftrightarrow \text{subgroups of } G$$

which sends any transitive  $G$ -set  $M$  equipped with a distinguished point  $r \in M$  to the stabilizer subgroup  $G_r = \{g \in G \mid g * r = r\}$ , and any subgroup  $H \subseteq G$  to the action of  $G$  on cosets  $G/H$  together with the distinguished coset  $H$ . In particular, every rooted trivalent map uniquely determines a subgroup of the modular group  $\mathcal{T} \cong \text{PSL}(2, \mathbb{Z})$ , while an unrooted map only determines one up to conjugacy [7, 23].

Finally, it is fairly common (cf. [7, 23]) to relax the conditions of fixed point-freeness in the definition of a general map and/or of a trivalent map. Intuitively, fixed points of  $e$  represent “dangling” edges, while fixed points of  $v$  represent univalent vertices. Precise formulations differ, however, and in Section 5 we will introduce a generalization of the classical definition of rooted trivalent maps which is motivated by the correspondence with linear lambda terms.

### 3 Basic definitions for linear lambda terms

Here we cover the small amount of background on lambda calculus that we will need in order to talk about linear lambda terms (for a more general introduction, see [1]). The terms of pure lambda calculus are constructed from variables  $(x, y, \dots)$  using only the two basic operations of *application*  $t(u)$  and *abstraction*  $\lambda x[t]$ . Within a given term, one distinguishes *free* variables from *bound* variables. An abstraction  $\lambda x[t]$  is said to bind the occurrences of  $x$  within the subterm  $t$ , and any variable which is not bound by an abstraction is said to be free. Two lambda terms are considered equivalent (“ $\alpha$ -equivalent”) if, roughly speaking, they differ only by renaming of bound variables.

A term is said to be *linear* if every variable (free or bound) has exactly one occurrence: for example, the terms  $\lambda x[x(\lambda y[y])]$ ,  $\lambda x[\lambda y[x(y)]]$ , and  $\lambda x[\lambda y[y(x)]]$  are linear, but the terms  $\lambda x[x(x)]$ ,  $\lambda x[\lambda y[x]]$ , and  $\lambda x[\lambda y[y]]$  are non-linear. To make this definition more precise, it is natural to consider linear lambda terms as indexed explicitly by lists of free variables, called *contexts*, which also affects the definition of  $\alpha$ -equivalence.

**Definition 3.1.** A **context** is an ordered list of distinct variables  $\Gamma = (x_1, \dots, x_k)$ . We write  $(\Gamma, \Delta)$  for the concatenation of two contexts  $\Gamma$  and  $\Delta$ , with the implicit condition that they contain disjoint sets of variables. Let  $\Gamma \vdash t$  be the relation between contexts and lambda terms defined inductively by the following rules:

$$\frac{}{x \vdash x} \quad \frac{\Gamma \vdash t \quad \Delta \vdash u}{\Gamma, \Delta \vdash t(u)} \quad \frac{\Gamma, x \vdash t}{\Gamma \vdash \lambda x[t]} \quad \frac{\Gamma, y, x, \Delta \vdash t}{\Gamma, x, y, \Delta \vdash t} \quad (1)$$

Then a **linear lambda term** is a pair  $(\Gamma, t)$  of a context and a term such that  $\Gamma \vdash t$ . Two linear lambda terms  $(\Gamma, t)$  and  $(\Gamma', t')$  are said to be  **$\alpha$ -equivalent** if they only differ by a series of changes of free or bound variables, defined as follows. Supposing that  $\Gamma = (\Gamma_1, x, \Gamma_2)$  and  $\Gamma' = (\Gamma_1, y, \Gamma_2)$ , then  $(\Gamma, t)$  and  $(\Gamma', t')$  differ by a single **change of free variable** if  $t' = t\{y/x\}$ , where  $t\{y/x\}$  denotes the substitution of  $y$  for  $x$  in  $t$ . Similarly,  $(\Gamma, t)$  and  $(\Gamma', t')$  differ by a single **change of bound variable** if  $t'$  arises from  $t$  by replacing some subterm  $\lambda x[u]$  by  $\lambda y[u\{y/x\}]$ .

For combinatorists, it might be helpful to see the two-variable generating function counting  $\alpha$ -equivalence classes of linear lambda terms of a given size in a given context. Let  $t_{n,k}$  stand for the number of  $\alpha$ -equivalence classes of linear lambda terms with  $n$  total applications and abstractions and with  $k$  free variables. Then the generating function  $L(z, x) = \sum_{n,k} t_{n,k} \frac{x^k z^n}{k!}$  satisfies the following functional-differential equation:

$$L(z, x) = x + zL(z, x)^2 + z \frac{\partial}{\partial x} L(z, x) \quad (2)$$

The three summands in equation (2) correspond to the three rules on the left side of (1),

$$\frac{}{x \vdash x} \quad \frac{\Gamma \vdash t \quad \Delta \vdash u}{\Gamma, \Delta \vdash t(u)} \quad \frac{\Gamma, x \vdash t}{\Gamma \vdash \lambda x[t]}$$

while the rule on the right side of (1)

$$\frac{\Gamma, y, x, \Delta \vdash t}{\Gamma, x, y, \Delta \vdash t}$$

explains why the generating function  $L(z, x)$  is of *exponential type* in the parameter  $x$ : if  $(\Gamma, t)$  is a linear lambda term, then so is  $(\Gamma', t)$  for any permutation  $\Gamma'$  of  $\Gamma$ . Finally, note that instantiating  $L(z, 0)$  gives the ordinary generating function counting *closed* linear lambda terms, and which also counts rooted trivalent maps (OEIS A062980).

Besides the notion of  $\alpha$ -equivalence itself, the real interest of lambda calculus is that one can *calculate* with it using the rules of  $\beta$ -reduction and/or  $\eta$ -expansion:

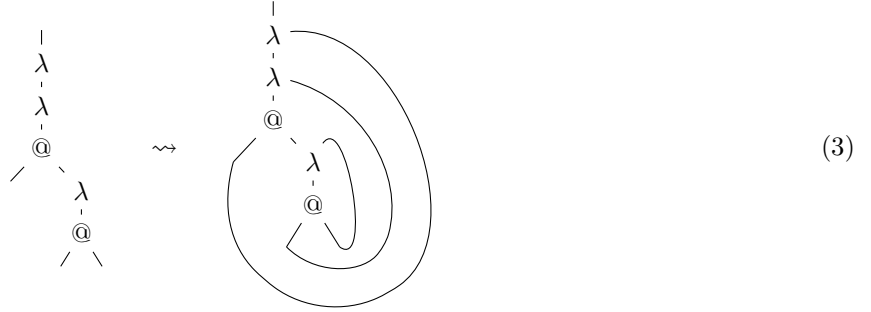
$$(\lambda x[t])(u) \xrightarrow{\beta} t\{u/x\} \quad t \xrightarrow{\eta} \lambda x[t(x)]$$

Since  $\beta$ -reduction and  $\eta$ -expansion preserve linearity, the fragment of lambda calculus consisting of the linear lambda terms can be seen as a proper subsystem with various special properties (for example, computing the  $\beta$ -normal form of a linear lambda term is PTIME-complete [13], whereas for non-linear terms it is undecidable whether a normal form even exists). Although it is sufficient to consider  $\alpha$ -equivalence for the purpose of presenting the bijection between linear lambda terms and rooted trivalent maps (which we will describe in Sections 5 and 6), we should nonetheless be aware that  $\beta$ -reduction and  $\eta$ -expansion are lurking in the background (and as mentioned in the introduction, there are some known connections between enumeration of  $\beta$ -normal terms and enumeration of rooted maps [24, 25]).

## 4 String diagrams for reflexive objects

A natural way of visualizing a lambda term is to begin by drawing a tree representing the underlying structure of applications and abstractions (what is called the *lambda skeleton* in [24]), and then add extra edges connecting each bound variable occurrence to its corresponding abstraction. For example, for  $\lambda x[\lambda y[x(\lambda z[y(z)])]]$

we begin with the tree on the left, adding links to obtain the diagram on the right:



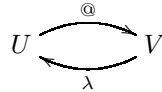
This approach is especially natural for linear lambda terms, since each  $\lambda$ -abstraction binds exactly one variable occurrence. Mairson [12] refers to these kinds of diagrams as *proof-nets* (probably because they are essentially equivalent to Girard’s proof-nets for the implicative fragment of linear logic), while Bodini et. al [3] speak of them as “syntactic trees”. I do not know just how far back the idea goes, but I’ve even seen such a diagram used to display a (linear) lambda term in an old essay by Knuth [10], who called it a particular kind of *information structure*.

In Sections 5 and 6 we will use these types of diagrams to help explain the bijection between linear lambda terms and rooted trivalent maps. The aim of this section is to briefly recount for the interested reader how such diagrams may be understood formally within the framework of *string diagrams* [8], as a notation for endomorphisms of a *reflexive object*. Some familiarity with the relevant categorical background will be assumed. (The analysis I give here follows [24, §3.1] but is considerably simplified.)

The key idea is to build on Dana Scott’s insight that the equational theory of pure lambda calculus can be modelled using a “reflexive object” in a cartesian closed category [16], and generalize it to the setting of symmetric monoidal closed (smc) bicategories [18]. We adopt the following definition:

**Definition 4.1.** A **reflexive object** in a smc bicategory  $\mathcal{K}$  is an object  $U$  equipped with an adjunction to its space of internal endomorphisms  $U \multimap U$ .

So, a reflexive object in  $\mathcal{K}$  consists of a pair of 1-cells



where the object  $V = U \multimap U$  comes with an equivalence of categories  $\mathcal{K}(X \otimes U, U) \cong \mathcal{K}(X, V)$  natural in  $X$ , together with a pair of 2-cells

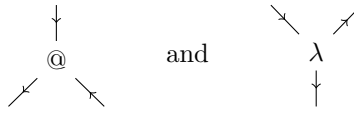


satisfying the zig-zag identities  $(\lambda\beta) \circ (\eta\lambda) = 1_\lambda$  and  $(\beta@) \circ (@\eta) = 1_@$ . A reflexive object provides a model of linear lambda calculus in the following sense (generalizing the classical situation [16, 5]):

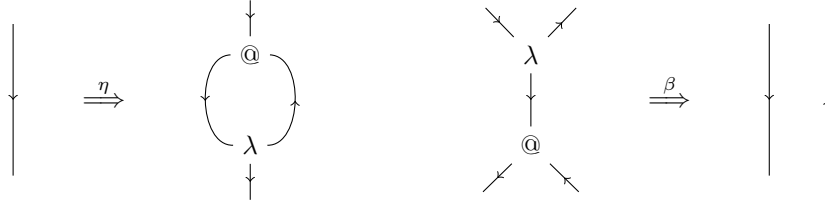
**Claim 4.2** (Soundness). *Let  $U$  be a reflexive object in a smc bicategory  $\mathcal{K}$ . Any linear lambda term  $(\Gamma, t)$  can be interpreted as a 1-cell  $\llbracket t \rrbracket : U^{\otimes k} \rightarrow U$  in  $\mathcal{K}$ , where  $\Gamma = x_1, \dots, x_k$  and  $U^{\otimes k}$  is the  $k$ -fold tensor product of  $U$ . Moreover, this interpretation respects  $\alpha$ -equivalence,  $\beta$ -reduction, and  $\eta$ -expansion.*

Now, a special kind of smc bicategory is a *compact closed* bicategory [18], where  $U \multimap U \cong U \otimes U^*$ . There is a fairly standard set of conventions for drawing morphisms of compact closed (bi)categories as string

diagrams [17, 18], typically using orientations on the “wires” to distinguish between an object  $A$  and its dual  $A^*$ . Applying these conventions to the data of a reflexive object in a compact closed bicategory, the 1-cells  $@ : U \rightarrow U \otimes U^*$  and  $\lambda : U \otimes U^* \rightarrow U$  get drawn (running down the page) as nodes of the shape

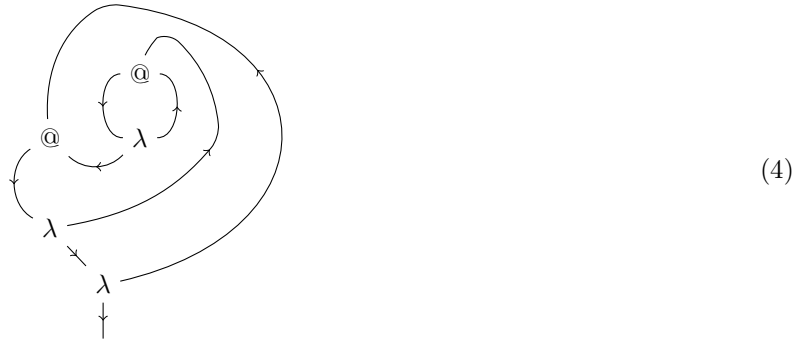


while the 2-cells  $\eta$  and  $\beta$  become rewriting rules



with the zig-zag identities expressing a coherence condition on these rewriting rules. For ease of reference, I’ll also give names (adopted from Mairson [12]) to the different wires positioned around the 1-cells: running clockwise around an @-node, the incoming wire at the top is called the **function** port, followed by the (incoming) **argument** and (outgoing) **continuation**, and running counterclockwise around a  $\lambda$ -node, the outgoing wire at the bottom is called the **root** port, followed by the (outgoing) **parameter** and (incoming) **body**.<sup>1</sup>

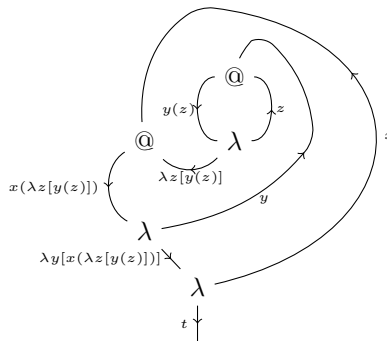
As an example, the closed linear lambda term  $t = \lambda x[\lambda y[x(\lambda z[y(z)])]]$  we considered above denotes a morphism  $[[t]] : 1 \rightarrow U$  in any smc bicategory with a reflexive object  $U$ . Drawing this 1-cell as a string diagram using the compact closed conventions we obtain the following picture:



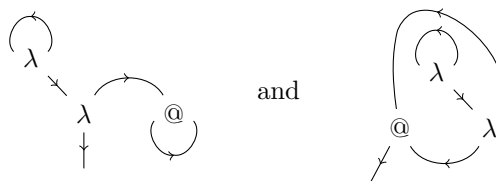
Observe that this diagram is essentially the same as the one in (3), just turned upside down and with explicit orientations on the wires. The correspondence with the original linear lambda term can be made a bit more

<sup>1</sup>I should point out that this way of ordering the wires corresponds to what was called the “RL” convention in [24, §3.1], rather than the “LR” convention which was mainly used in that paper.

evident by labelling the wires with subterms of  $t$ :



One thing it is important to point out is that not every physical combination of @-nodes and  $\lambda$ -nodes represents a linear lambda term. For instance, the diagrams



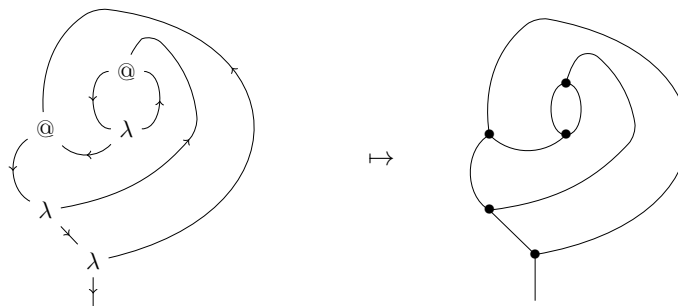
do not correspond to the interpretation of a linear lambda term, although they do represent morphisms  $1 \rightarrow U$  in any *compact closed* bicategory with a reflexive object  $U$ . Nonetheless, the interpretation of linear lambda terms using reflexive objects in smc bicategories is complete in the following sense:

**Claim 4.3** (Completeness). *There is a smc bicategory  $\mathcal{K}_\Lambda$  equipped with a reflexive object  $U$ , such that every 1-cell  $f : U^{\otimes k} \rightarrow U$  is the interpretation  $f = \llbracket t \rrbracket$  of a unique linear lambda term  $t$  with  $k$  free variables, and such that there is a 2-cell  $\llbracket t_1 \rrbracket \Rightarrow \llbracket t_2 \rrbracket$  if and only if  $t_1$  is  $\beta\eta$ -convertible to  $t_2$ .*

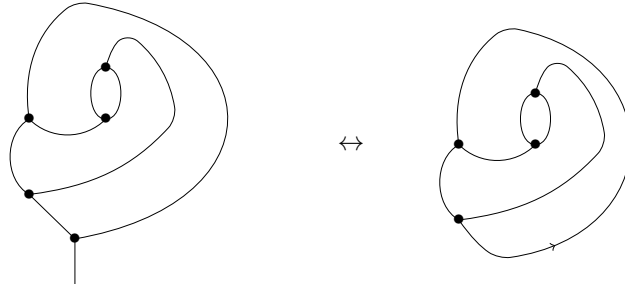
The proof essentially follows Hyland's analysis [5] of Scott's Representation Theorem, replacing cartesian closed categories by smc bicategories. The idea is to take  $\mathcal{K}_\Lambda$  as a presheaf bicategory  $[\mathcal{C}^{\text{op}}, \mathbf{Cat}]$ , where  $\mathcal{C}$  is a symmetric monoidal bicategory whose 0-cells are contexts, 1-cells are tuples of linear lambda terms, and 2-cells are rewritings between tuples. The smc structure on  $[\mathcal{C}^{\text{op}}, \mathbf{Cat}]$  is defined by Day convolution, and the reflexive object is constructed as the representable presheaf for a singleton context.

## 5 From linear lambda terms to rooted trivalent maps

Once we view linear lambda calculus through the lens of string diagrams, it is pretty clear how to turn any closed linear lambda term into a rooted trivalent map: just look at its string diagram and forget the distinction between @-nodes and  $\lambda$ -nodes, as well as the orientations on the wires. Here we apply this transformation on  $\lambda x[\lambda y[x(\lambda z[y(z)])]]$  and its corresponding string diagram (4):



Even though we identify @-nodes and  $\lambda$ -nodes, it is important that we take care to remember the *ordering* of the wires around each node, since we are interested in obtaining a *map* rather than an abstract trivalent graph. Strictly speaking the diagram on the right is not a trivalent map (in the sense of Defn. 2.3) since it has a dangling edge, but it can be interpreted as a rooted trivalent map (in the sense of Defn. 2.4) by reading the outgoing trivalent vertex as a “normal vector” to the root dart of a map with that vertex smoothed out:



Actually there is a hiccup in performing this last step for the identity term  $I = \lambda x[x]$ ,

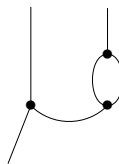


since the no-vertex map is not technically a rooted map (again in the sense of Defn. 2.4, although studies of the combinatorics of rooted maps often treat the empty map as an exceptional case [21]). This hiccup will soon go away, however, once we consider a more general notion of rooted trivalent map.

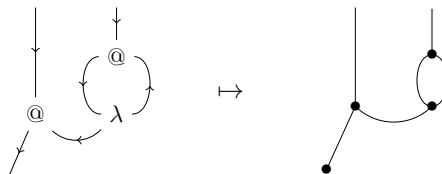
The real reason for considering this more general notion is that we would also like to interpret linear lambda terms with free variables as rooted trivalent maps. Consider the term  $x(\lambda z[y(z)])$  with free variables  $x$  and  $y$ , whose string diagram corresponds to a subdiagram of (4):



Identifying @-nodes and  $\lambda$ -nodes in (5) yields a trivalent map with three dangling edges,



but since it is no longer clear from the diagram which dangling edge marks the root, we attach an extra univalent vertex to one of them (turning it into a full edge):



By considering the output of this transformation on linear lambda terms with any number of free variables we arrive at the following generalization of Defn. 2.4:



**Definition 5.1.** For any  $G$ -set  $X$  let  $\text{fix}_g(X) \stackrel{\text{def}}{=} \{x \in X \mid g*x = x\}$ , and again take  $\mathcal{T} \stackrel{\text{def}}{=} \langle v, e \mid v^3 = e^2 = 1 \rangle$ . A **rooted trivalent map with boundary** is a transitive  $\mathcal{T}$ -set  $M$  equipped with a distinguished element  $r \in M$  and a list of distinct elements  $x_1, \dots, x_k \in M$ , such that  $\text{fix}_v(M) = \{r\}$  and  $\text{fix}_e(M) = \{x_1, \dots, x_k\}$ . We refer to the unique  $v$ -fixed point as the **root**  $r(M)$  of the map, the ordered list of  $e$ -fixed points as the **boundary**  $\Gamma(M)$  of the map, and to the integer  $k$  (= the number of  $e$ -fixed points) as the **degree** of the boundary. A rooted trivalent map with boundary of degree 0 is called a **closed** rooted trivalent map.

From now on, when we say “rooted trivalent map” without qualification we mean rooted trivalent map with boundary in the sense of Defn. 5.1, referring to the sense of Defn. 2.4 as “classical rooted trivalent map”.

**Proposition 5.2.** For all  $n > 0$ , there is a bijection between closed rooted trivalent maps with  $n+1$  trivalent vertices and classical rooted trivalent maps with  $n$  trivalent vertices. This extends to a bijection for all  $n \geq 0$  if the empty  $\mathcal{T}$ -set is admitted as a classical rooted trivalent map.

*Proof.* As explained near the beginning of this section. □

Observe that the simplest possible rooted trivalent map with boundary is the singleton  $\mathcal{T}$ -set  $M = \{x\}$  with  $r(M) = \Gamma(M) = x$ , corresponding to the trivial map  $\downarrow$  with no trivalent vertices and one free edge. With this definition, it is clear how linear lambda term induces a rooted trivalent map with boundary.

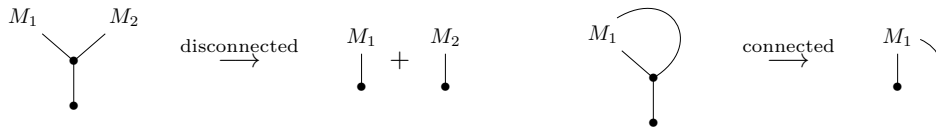
**Proposition 5.3.** To any linear lambda term with  $k$  free variables,  $p$  applications and  $q$  abstractions there is naturally associated a rooted trivalent map with boundary of degree  $k$  and  $p+q$  trivalent vertices.

*Proof.* Consider the string diagram of the term, which has  $k$  incoming wires and one outgoing wire, as well as  $p$  @-nodes and  $q$   $\lambda$ -nodes internal to the diagram. Transform @-nodes and  $\lambda$ -nodes into trivalent vertices, attach a univalent vertex to the end of the outgoing wire, and finally forget the orientations of the wires. The result is manifestly a rooted trivalent map with boundary of degree  $k$  and  $p+q$  trivalent vertices. □

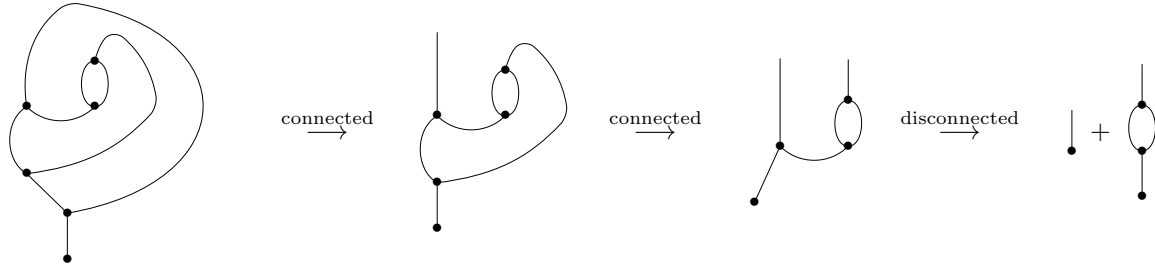
We call the map described in the proof of Prop. 5.3 the **underlying rooted trivalent map** of a linear lambda term. Although the high-level description is clear enough, we can also explicitly compute the permutations  $v$  and  $e$  associated to the underlying rooted trivalent map by induction on the structure of the linear lambda term, which just requires a bit of bookkeeping. What is somewhat more surprising is that this “forgetful” transformation can be reversed – that is the topic of the next section.

## 6 From rooted trivalent maps to linear lambda terms

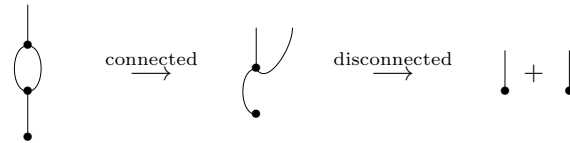
Given a rooted trivalent map  $M$  (with a boundary of dangling edges  $\Gamma(M)$ ), our task is to find a linear lambda term (with free variables  $\Gamma$ ) whose underlying rooted trivalent map is  $M$  (hopefully, such a linear lambda term always exists and is unique). To be able to do this, clearly we will somehow have to decide for each trivalent vertex whether it corresponds to an application (@) or an abstraction ( $\lambda$ ). The trick is that there is always a unique answer for the trivalent vertex incident to the root: if removing that vertex disconnects the underlying graph then the vertex corresponds to an @-node, and otherwise it corresponds to a  $\lambda$ -node. In either case, we can re-root the resulting submap(s) (while adjusting the boundary)



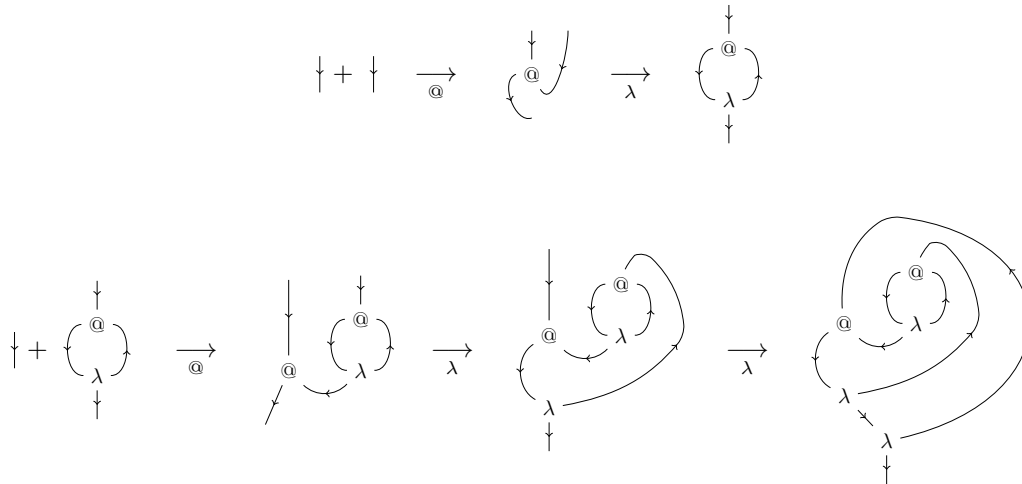
and continue the process recursively until we eventually arrive at the trivial map  $\downarrow$ . For example, here we start applying this process to a closed rooted trivalent map



and continue until we are left with only trivial maps:



From the trace of this decomposition, we can reconstruct the necessarily unique linear lambda term whose underlying rooted trivalent map is our original map:

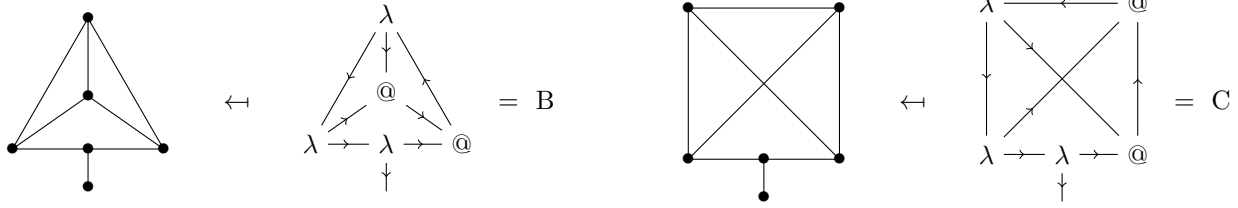


**Theorem 6.1.** *To any rooted trivalent map  $M$  with boundary of degree  $k$  and  $n$  trivalent vertices there is a unique linear lambda term with  $k$  free variables,  $p$  applications and  $q$  abstractions whose underlying rooted trivalent map is  $M$ , for some  $p + q = n$ .*

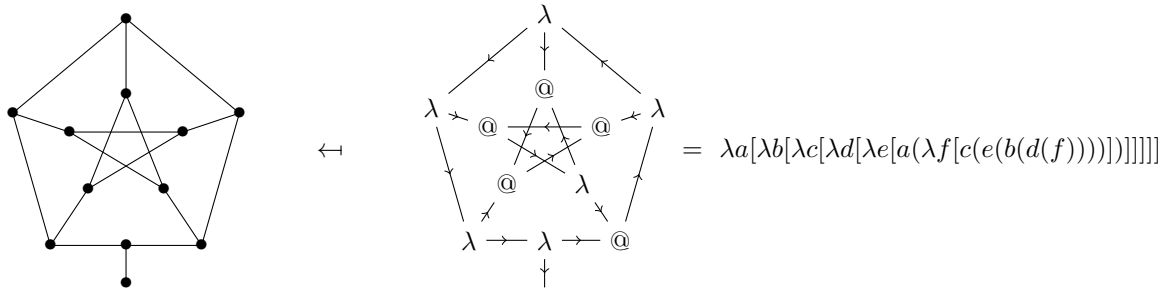
*Proof.* As sketched above, by induction on  $n$ . Note that uniqueness relies on the fact that we define rooted trivalent maps with boundary as equipped with a fixed ordering on dangling edges, which determines the ordering of the free variables inside the context of the corresponding linear lambda term.  $\square$

**Corollary 6.2.** *Rooted trivalent maps with boundary of degree  $k$  and  $n$  trivalent vertices are in one-to-one correspondence with linear lambda terms with  $k$  free variables and  $n$  total applications and abstractions. Both families are counted by the generating function satisfying the functional-differential equation (2).*

**Example 1.** The standard combinators [4]  $B = \lambda x[\lambda y[\lambda z[x(yz)]]]$  and  $C = \lambda x[\lambda y[\lambda z[(xz)y]]]$  correspond to two different rooted embeddings of the  $K_4$  graph (respectively, a planar embedding and a toric one):

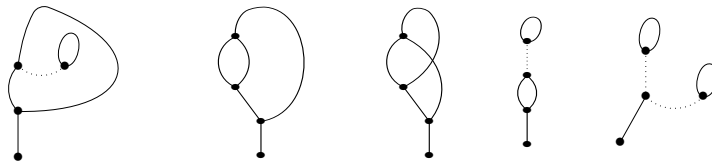


**Example 2.** A rooted embedding of the Petersen graph, and its corresponding linear lambda term:

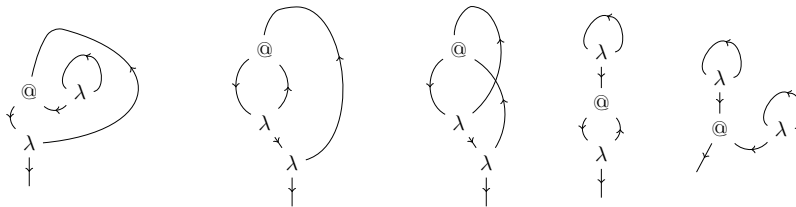


## 7 Indecomposable linear lambda terms and the 4CT

Recall that a *bridge* in a connected graph is any edge whose removal disconnects the graph. Among the first five non-trivial closed rooted trivalent maps, exactly three of them contain bridges (we do not count the outgoing root edge as a bridge):



If we look at the corresponding string diagrams,



we see that each of the bridges corresponds to a wire oriented towards an @-node (either in function or in argument position), and that it sends a *closed subterm* of the underlying linear lambda term (in these three cases an identity term  $I$ ) to that @-node.

**Definition 7.1.** Let  $(\Gamma, t)$  be a linear lambda term. A **subterm** of  $(\Gamma, t)$  is a linear lambda term  $(\Delta, u)$  that appears in the derivation of  $\Gamma \vdash t$ . Explicitly:

- $(\Gamma, t)$  is a subterm of itself;

- if  $t = t_1(t_2)$  for some  $\Gamma_1 \vdash t_1$  and  $\Gamma_2 \vdash t_2$ , then every subterm  $(\Delta, u)$  of  $(\Gamma_1, t_1)$  or  $(\Gamma_2, t_2)$  is also a subterm of  $(\Gamma, t)$ ; and
- if  $t = \lambda x.t_1$  for some  $\Gamma, x \vdash t_1$ , then every subterm  $(\Delta, u)$  of  $(\Gamma, x), t_1)$  is also a subterm of  $(\Gamma, t)$ .

We refer to all the subterms of  $(\Gamma, t)$  other than  $(\Gamma, t)$  itself as **proper subterms**.

**Definition 7.2.** A linear lambda term is said to be **decomposable** if it has a closed proper subterm, and **indecomposable** otherwise.

**Claim 7.3.** A closed rooted trivalent map is bridgeless if and only if the corresponding closed linear lambda term is indecomposable.

To prove this claim, let's first recall the notion of the *lambda lifting* of a term with free variables.

**Definition 7.4.** Let  $(\Gamma, t)$  be a linear lambda term, where  $\Gamma = (x_1, \dots, x_k)$ . The **lambda lifting** of  $(\Gamma, t)$  is the closed linear lambda term  $\lambda\Gamma[t] \stackrel{\text{def}}{=} \lambda x_1[\dots \lambda x_k[t] \dots]$ .

*Proof of Claim 7.3.* By Thm. 6.1, it suffices to do an induction over linear lambda terms, and check whether the underlying rooted trivalent map of their lambda lifting contains a bridge (and again, we do not count the outgoing root edge itself as a bridge). From examination of the root-deletion procedure, it is immediate that the only way of potentially introducing a bridge is by using an @-node to form an application  $t = t_1(t_2)$ , so we just have to check whether or not removing the edge corresponding to either the function port ( $t_1$ ) or argument port ( $t_2$ ) of the @-node disconnects the diagram of  $\lambda\Gamma[t]$ . Well, if  $t_i$  has a free variable  $x$ , then the subdiagram rooted at  $t_i$  will remain connected to the root port of  $\lambda\Gamma[t]$ , by a path running through the root port of the  $\lambda$ -node for  $x$ . Hence, the edge corresponding to  $t_i$  is a bridge in the underlying rooted trivalent map of  $\lambda\Gamma[t]$  just in case  $t_i$  is closed.  $\square$

This analysis immediately suggests a way of enumerating bridgeless rooted trivalent maps.

**Proposition 7.5.** The generating function  $L_{\text{ind}}(z, x)$  counting indecomposable linear lambda terms by size (= number of applications and abstractions) and number of free variables satisfies the following functional-differential equation:

$$L_{\text{ind}}(z, x) = x + z(L_{\text{ind}}(z, x) - L_{\text{ind}}(z, 0))^2 + z \frac{\partial}{\partial x} L_{\text{ind}}(z, x) \quad (6)$$

In particular, the OGF

$$L_{\text{ind}}(z, 0) = z + 2z^3 + 20z^5 + 352z^7 + 8624z^9 + 266784z^{11} + \dots$$

counts closed indecomposable linear lambda terms by size, as well as closed bridgeless rooted trivalent maps (on oriented surfaces of arbitrary genus) by number of trivalent vertices.

Now, let us say that a linear lambda term is planar just in case its underlying rooted trivalent map is planar. Planar lambda terms have the special property that after we've fixed the convention for ordering wires around @-nodes and  $\lambda$ -nodes, there is always exactly one planar term with any given underlying (unary-binary) tree of applications and abstractions (see [24, §2]). This makes them easy to count, and the following "discrete analogues" of (2) and (6) define the two-variable generating functions for planar lambda terms and indecomposable planar lambda terms, respectively:

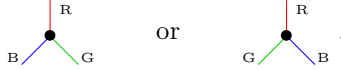
$$P(z, x) = x + zP(z, x)^2 + z \frac{P(z, x) - P(z, 0)}{x} \quad (7)$$

$$P_{\text{ind}}(z, x) = x + z(P_{\text{ind}}(z, x) - P_{\text{ind}}(z, 0))^2 + z \frac{P_{\text{ind}}(z, x) - P_{\text{ind}}(z, 0)}{x} \quad (8)$$

In particular, the OGF  $P_{\text{ind}}(z, 0) = z + z^3 + 4z^5 + 24z^7 + 176z^9 + 1456z^{11} + \dots$  counts rooted bridgeless planar trivalent maps by number of trivalent vertices, as originally enumerated by Tutte [20] (OEIS A000309;

keep in mind that we define closed rooted trivalent maps to contain one extra trivalent vertex relative to the classical definition, cf. Prop. 5.2).

Bridgeless planar trivalent maps are closely related to the Four Color Theorem: by Tait's well-known reduction [19], the statement that every bridgeless planar map has a proper 4-coloring of its faces is equivalent to the statement that every bridgeless planar trivalent map has a proper 3-coloring of its edges, i.e., a labelling of the edges by colors in  $\{R, G, B\}$  such that every vertex has the form



For the purposes of coloring, there is little difference between rooted and unrooted maps: without loss of generality, it suffices to root a trivalent map  $M$  arbitrarily at some edge by splitting it with a trivalent vertex, assign both halves of that edge the same arbitrary color ( $\begin{array}{c} \text{R} \\ \text{---} \\ \text{M} \\ \text{---} \\ \text{R} \end{array}$ ), and then look for a proper 3-coloring of the remaining edges.

It seems that the problem of 3-coloring the edges of a rooted trivalent map may be naturally formulated as a *typing* problem in linear lambda calculus. Typing for linear lambda calculus is standardly defined by the following rules:

$$\frac{}{x : X \vdash x : X} \quad \frac{\Gamma \vdash t : X \multimap Y \quad \Delta \vdash u : X}{\Gamma, \Delta \vdash t(u) : Y} \quad \frac{\Gamma, x : X \vdash t : Y}{\Gamma \vdash \lambda x[t] : X \multimap Y} \quad \frac{\Gamma, y : Y, x : X, \Delta \vdash t : Z}{\Gamma, x : X, y : Y, \Delta \vdash t : Z} \quad (9)$$

General types  $(X, Y, \dots)$  are built up from some set of type variables  $(\alpha, \beta, \dots)$  using only implication  $(X \multimap Y)$ , and the typing judgment  $x_1 : X_1, \dots, x_k : X_k \vdash t : Y$  expresses that the given term  $t$  has type  $Y$  assuming that the free variables have the prescribed types  $X_1, \dots, X_k$ . In this way, closed linear lambda terms can be seen as proofs of tautologies in a very weak, purely implicative logic (sometimes called BCI logic, after the combinators B, C, and I). Moreover, planar lambda terms are typable without using the rightmost rule, resulting in an even weaker logic.

The standard typing rules can also be expressed (more concisely) using string diagrams, where they correspond to the following conditions for annotating the wires by types (cf. [12, 26]):



In this form, the connection to edge-coloring is more suggestive. Indeed, we can use a specific interpretation of types in order to obtain another reformulation of the map coloring theorem (cf. [15, 9, 2]).

**Definition 7.6.** Let  $\mathbb{T} = \{R, G, B, 1\}$ , and define an operation  $\multimap : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$  by

$$\begin{array}{llll} B \multimap R = G & B \multimap B = 1 & B \multimap G = R & 1 \multimap X = X \\ G \multimap R = B & G \multimap B = R & G \multimap G = 1 & X \multimap 1 = X \\ R \multimap R = 1 & R \multimap B = G & R \multimap G = B & \end{array}$$

We write  $\vdash_{\mathbb{T}}$  for the typing judgment induced from (9) by restricting types to elements of  $\mathbb{T}$  and interpreting  $\multimap$  as described. A **3-typing** of a linear lambda term  $t$  with free variables  $x_1, \dots, x_k$  is defined as a derivation of the typing judgment  $x_1 : X_1, \dots, x_k : X_k \vdash_{\mathbb{T}} t : Y$  for some  $X_1, \dots, X_k$  and  $Y$  in  $\mathbb{T}$ . The 3-typing is said to be **proper** if no proper subterm of  $t$  is assigned the type 1.

**Claim 7.7** (Reformulation of 4CT). *Every planar indecomposable linear lambda term has a proper 3-typing.*

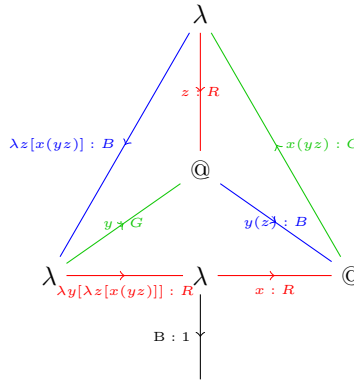
**Example 3.** The B combinator (Example 1) has most general type

$$B : (\beta \multimap \gamma) \multimap ((\alpha \multimap \beta) \multimap (\alpha \multimap \gamma))$$

corresponding to the following formal typing derivation with type variables  $\alpha, \beta, \gamma$ :

$$\frac{\frac{\frac{\frac{\frac{\frac{}{y : \alpha \multimap \beta \vdash y : \alpha \multimap \beta}{} \quad \frac{}{z : \alpha \vdash z : \alpha}}{}{y : \alpha \multimap \beta, z : \alpha \vdash y(z) : \beta}}{x : \beta \multimap \gamma \vdash x : \beta \multimap \gamma}}{x : \alpha \multimap \beta, y : \alpha \multimap \beta, z : \alpha \vdash x(yz) : \gamma}}{x : \alpha \multimap \beta, y : \alpha \multimap \beta \vdash \lambda z[x(yz)] : \alpha \multimap \gamma}}{x : \beta \multimap \gamma \vdash \lambda y[\lambda z[x(yz)]] : (\alpha \multimap \beta) \multimap (\alpha \multimap \gamma)}}{\vdash \lambda x[\lambda y[\lambda z[x(yz)]]] : (\beta \multimap \gamma) \multimap ((\alpha \multimap \beta) \multimap (\alpha \multimap \gamma))}}$$

Instantiating  $\alpha = R$ ,  $\beta = B$ , and  $\gamma = G$ , we obtain a proper 3-typing of the combinator:



**Acknowledgments.** I am grateful to Alain Giorgetti for introducing me to the idea of Tutte decomposition and for emphasizing its importance as a way of representing rooted maps. Special thanks to Andrej Bauer for inviting me to give a talk at the University of Ljubljana Foundations Seminar, which was the impetus for writing up these notes. Thanks also to Doron Zeilberger for helpful comments on an earlier draft. Finally, this work has been funded by the ERC Advanced Grant ProofCert.

## References

- [1] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic 103, second, revised edition, North-Holland, Amsterdam, 1984.
- [2] Dror Bar-Natan. Lie algebras and the four color theorem, *Combinatorica* 17, 43–52, 1997.
- [3] O. Bodini, D. Gardy, and A. Jacquot. Asymptotics and random sampling for BCI and BCK lambda terms. *Theoretical Computer Science*, 502:227–238, 2013.
- [4] H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic*, vol. II, North-Holland, 1972.
- [5] Martin Hyland. Classical lambda calculus in modern dress. To appear in *Mathematical Structures in Computer Science*. arXiv:1211.5762
- [6] Gareth A. Jones and David Singerman. Theory of maps on orientable surfaces. *Proceedings of the London Mathematical Society*, 37:273–307, 1978.

- [7] Gareth A. Jones and David Singerman. Maps, hypermaps, and triangle groups. In *The Grothendieck Theory of Dessins d'Enfants*, L. Schneps (ed.), London Mathematical Society Lecture Note Series 200, Cambridge University Press, 1994.
- [8] André Joyal and Ross Street. The geometry of tensor calculus I. *Advances in Mathematics*, 102:20–78, 1993.
- [9] Louis H. Kauffman. Map Coloring and the Vector Cross Product. *Journal of Combinatorial Theory B* 48:145–154, 1990.
- [10] Donald E. Knuth. Examples of formal semantics. In *Symposium on Semantics of Algorithmic Languages*, E. Engeler (ed.), Lecture Notes in Mathematics 188, Springer, 1970.
- [11] Sergei K. Lando and Alexander K. Zvonkin. *Graphs on Surfaces and Their Applications*, Encyclopaedia of Mathematical Sciences 141, Springer-Verlag, 2004.
- [12] Harry G. Mairson. From Hilbert Spaces to Dilbert Spaces: Context Semantics Made Simple. In *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science*, 2–17, Kanpur, India, 2002.
- [13] Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming*, 14:6, 2004.
- [14] OEIS Foundation Inc., The On-Line Encyclopedia of Integer Sequences.
- [15] Roger Penrose. Applications of Negative Dimensional Tensors. In D. Welsh (ed.), *Combinatorial Mathematics and its Application*, 221–243, 1971.
- [16] Dana S. Scott. Relating theories of the  $\lambda$ -calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism* (eds. Hindley and Seldin), Academic Press, 403–450, 1980.
- [17] Peter Selinger. A survey of graphical languages for monoidal categories. In *New Structures for Physics* (ed. Bob Coecke), Springer Lecture Notes in Physics 813, 289–355, 2011.
- [18] Michael Stay. Compact closed bicategories. arXiv:1301.1053, 2013.
- [19] Robin Thomas. An Update on the Four-Color Theorem. *Notices of the American Mathematical Society* 45:7, 848–859, 1998.
- [20] W. T. Tutte. A census of Hamiltonian polygons. *Canadian Journal of Mathematics*, 14:402–417, 1962.
- [21] W. T. Tutte. On the enumeration of planar maps. *Bulletin of the American Mathematical Society*, 74:64–74, 1968.
- [22] W. T. Tutte. *Graph Theory as I Have Known it*. Oxford, 1998.
- [23] Samuel Vidal. *Groupe Modulaire et Cartes Combinatoires: Génération et Comptage*. PhD thesis, Université Lille I, France, July 2010.
- [24] Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Logical Methods in Computer Science*, 11(3:22):1–39, 2015.
- [25] Noam Zeilberger. Counting isomorphism classes of  $\beta$ -normal linear lambda terms. Submitted for publication. September 25, 2015. arXiv:1509.07596
- [26] Noam Zeilberger. Balanced polymorphism and linear lambda calculus. Talk at TYPES 2015, Tallinn, Estonia, 18 May 2015. Manuscript in preparation.