A correspondence between rooted planar maps and normal planar lambda terms

Zeilberger, Noam; Giorgetti, Alain

DOI:
10.2168/LMCS-11(3:22)2015

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Zeilberger, N & Giorgetti, A 2015, 'A correspondence between rooted planar maps and normal planar lambda terms', *Logical Methods in Computer Science*, vol. 11, no. 3, pp. 1-39. https://doi.org/10.2168/LMCS-11(3:22)2015

Logical Methods in Computer Science
Vol. 11(3:22)2015, pp. 1–39
www.lmcs-online.org

Submitted Aug. 28, 2014
Published Sep. 25, 2015

A CORRESPONDENCE BETWEEN ROOTED PLANAR MAPS AND NORMAL PLANAR LAMBDA TERMS

NOAM ZEILBERGER [a] AND ALAIN GIORGETTI [b]

[a] MSR-Inria, Joint Centre, 91120 Palaiseau, France
e-mail address: noam.zeilberger@gmail.com

[b] FEMTO-ST institute (UMR CNRS 6174 - UBFC/UFC/ENSMM/UTBM), University of Franche-Comté, 25030 Besançon, France / CASSIS project, Inria, 54600 Villers-les-Nancy, France
e-mail address: alain.giorgetti@femto-st.fr

Abstract. A *rooted planar map* is a connected graph embedded in the 2-sphere, with one edge marked and assigned an orientation. A term of the pure lambda calculus is said to be *linear* if every variable is used exactly once, *normal* if it contains no β-redexes, and *planar* if it is linear and the use of variables moreover follows a deterministic stack discipline. We begin by showing that the sequence counting normal planar lambda terms by a natural notion of size coincides with the sequence (originally computed by Tutte) counting rooted planar maps by number of edges. Next, we explain how to apply the machinery of string diagrams to derive a graphical language for normal planar lambda terms, extracted from the semantics of linear lambda calculus in symmetric monoidal closed categories equipped with a *linear reflexive object* or a *linear reflexive pair*. Finally, our main result is a size-preserving bijection between rooted planar maps and normal planar lambda terms, which we establish by explaining how *Tutte decomposition* of rooted planar maps (into vertex maps, maps with an isthmic root, and maps with a non-isthmic root) may be naturally replayed in linear lambda calculus, as certain surgeries on the string diagrams of normal planar lambda terms.

1. Introduction: a curious correspondence

The pure lambda calculus is a universal programming language based on only two primitive operations: for any pair of terms t and u, there is a term $t(u)$ representing the *application* of t to u, while for any pair of a term t and a variable x, there is a term $\lambda x.t$ representing the *abstraction* of t in x. Terms are always considered up to renaming of abstracted variables, so that for example $\lambda x.x$ and $\lambda y.y$ both represent the same term (intuitively standing for the identity function). The main source of computation is the rule of *β-reduction*:

$$(\lambda x.t)(u) \to t[u/x]$$

2012 ACM CCS: [**Mathematics of computing**]: Discrete mathematics—Combinatorics—Enumeration; Discrete mathematics—Graph theory—Graphs and surfaces; [**Theory of computation**]: Models of computation—Computability—Lambda calculus; Logic—Linear logic.

Key words and phrases: lambda calculus, combinatorics, string diagrams, rooted maps, planarity.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(3:22)2015

Here $t[u/x]$ denotes the *substitution* of u for x in t (which technically must be "capture-avoiding" in a sense we need not get into here [4]). Equating terms modulo β-reduction yields a theory known as Λ_β, wherein, for example, we can derive

$$(\lambda x.xx)(\lambda y.y) = (\lambda y.y)(\lambda y.y) = \lambda y.y \qquad \text{and} \qquad ((\lambda x.\lambda y.x)w)(\lambda z.z) = (\lambda y.w)(\lambda z.z) = w.$$

Although it is remarkable that such a simple and conceptual language is Turing-complete, the focus of this paper will be on a much more restrictive but still important subset of lambda calculus known as the *linear* fragment, defined by the requirement that every abstracted variable must be used exactly once. For example, all of the terms

$$\lambda x.\lambda y.yx \qquad \lambda x.x(\lambda y.y) \qquad \lambda x.\lambda y.xy$$

are linear, but all of the terms

$$\lambda x.xx \qquad \lambda x.\lambda y.x \qquad \lambda x.\lambda y.y$$

are non-linear. As one example of the special properties of the linear fragment, note that the problem of computing the β-normal form of a linear lambda term is PTIME-complete [21].

Among the linear terms, it is possible to identify an even more restrictive subset of terms which are *planar*, in the sense that (reading left-to-right) variables are used in the reverse order which they are abstracted. Thus the two linear terms

$$\lambda x.\lambda y.yx \qquad \lambda x.x(\lambda y.y)$$

are planar, but the linear term

$$\lambda x.\lambda y.xy$$

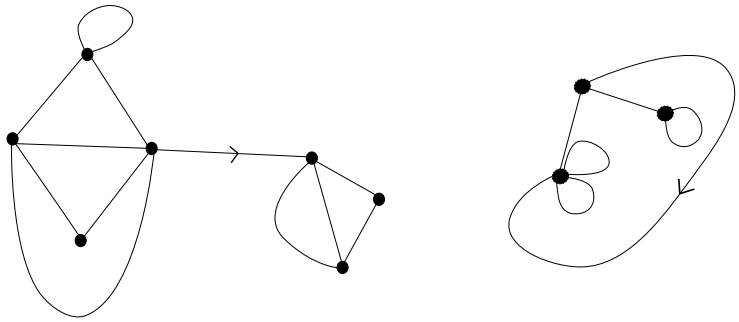
is non-planar.

Motivated by questions related to the study of *type refinement* [23], the first author of this paper counted β-normal planar lambda terms along a natural notion of size, and obtained the following sequence of first numbers through a simple recurrence equation:

$$1, 2, 9, 54, 378, 2916, 24057$$

Surprisingly, this sequence already existed in the Online Encyclopedia of Integer Sequences [24], corresponding to the first entries of a series which is indexed in the OEIS as **A000168**.

It turns out that this series is well-known in combinatorics (see, e.g., [9, VII.8.2]), and counts *rooted planar maps* by number of edges. A rooted planar map is essentially a connected graph drawn on the sphere with no crossing-edges, and with one edge marked and assigned an orientation. Here are two small examples, where we have chosen to project the maps onto the page so that the "infinite" (outer) face is to the left of the oriented root edge:



Rooted planar maps were originally enumerated in the 1960s by Tutte [33, 34] as part of an attack on the four-color theorem (which, of course, is about unrooted planar maps). What makes rooted maps easier to count than unrooted maps is that the latter can have non-trivial symmetries but the former cannot. Tutte was even able to derive a closed form for the total number of rooted planar maps with n edges: $\frac{2\cdot(2n)!\cdot 3^n}{n!\cdot(n+2)!}$.

The main result of this paper is a size-preserving bijection between rooted planar maps and normal planar lambda terms. We work towards this result as follows:

- In Section 2, we introduce linear lambda calculus from a combinatorial perspective, defining linear lambda terms as certain decorations of "lambda skeletons". We then give inductive definitions of various properties of linear lambda terms, and use these to derive functional equations for the generating functions counting normal (and "neutral") planar terms by size and number of free variables. By solving these equations, we demonstrate in particular that the sequence counting closed normal planar lambda terms indeed coincides with A000168.
- In Section 3, we explain how to apply the machinery of string diagrams to derive a graphical language for normal planar lambda terms. Our first step is a rational reconstruction of the well-known "lambda-graphs", as string diagrams extracted from the semantics of linear lambda calculus in symmetric monoidal closed categories equipped with a *linear reflexive object*. We then introduce the concept of a *linear reflexive pair* as a refinement of linear reflexive object, and use this to extract a coloring protocol for the string diagrams representing normal linear lambda terms.
- Finally, in Section 4 we give the size-preserving bijection between rooted planar maps and normal planar lambda terms. The idea of the bijection is based on Tutte's analysis of rooted planar maps now known as *Tutte decomposition* [34], which starts by establishing a trichotomy on rooted planar maps as either being the degenerate vertex map (with no edges), or else having an isthmic root, or else having a non-isthmic root. After giving a review of Tutte decomposition, we explain how his analysis may be naturally replayed in linear lambda calculus as certain surgeries on the string diagrams of normal planar lambda terms, and show how to use this to obtain a size-preserving bijection.

2. Lambda skeletons, planarity, neutral and normal terms

The main objects we study in this paper are lambda terms satisfying a combination of three properties: linearity, planarity, and absence of β-redexes. Since this is just a small fragment of lambda calculus, we do not need to introduce the full machinery of classical lambda calculus (for which the reader can see [4]), and instead take an approach inspired by the operadic perspective advocated by Hyland [14] since it makes the underlying combinatorial structure of linear/planar lambda terms more apparent.

We begin by defining "skeletons" of lambda terms, standing for lambda terms with placeholders for variable names.

Definition 2.1. A *lambda skeleton* is an element of the set $\mathcal{S}(i)$, defined for $i \in \mathbb{N}$ as the least graded set satisfying the following rules:

$$\frac{}{\textvisiblespace \in \mathcal{S}(1)}\ V \qquad \frac{p \in \mathcal{S}(j) \quad q \in \mathcal{S}(k)}{p(q) \in \mathcal{S}(j+k)}\ A \qquad \frac{p \in \mathcal{S}(i+1)}{\lambda\textvisiblespace.p \in \mathcal{S}(i)}\ L$$

The *degree* of a lambda skeleton $p \in \mathcal{S}(i)$ is the index i.

It is worth remarking that lambda skeletons are simply *unary-binary trees* (also known as *Motzkin trees*), where unary "L-nodes" stand for lambdas, binary "A-nodes" stand for applications, and "(V-)leaves" stand for variables. But whereas unary-binary trees are usually parameterized by total number of nodes (which gives rise to the *Motzkin numbers*, as the number of unary-binary trees with a given number of nodes [9, I.39]), lambda skeletons are parameterized by the *difference* between the number of leaves and the number of L-nodes. In particular, all of the sets $\mathcal{S}(i)$ are infinite, so lambda skeletons cannot be counted directly along their degree.

Linear lambda terms will be defined as certain *decorations* of lambda skeletons.

Definition 2.2. A *pseudo lambda term* is a lambda skeleton in which all occurrences of "$\llcorner$" have been replaced by variable names. Given a lambda skeleton $p \in \mathcal{S}(i)$, a list of variable names $\Gamma = x_1, \ldots, x_i$, and a pseudo lambda term t, we write $[\Gamma]t \in \Lambda_1(p)$ to indicate that t is a *linear lambda term* (with free variables Γ) *decorating* p, as defined by the following rules:

$$\frac{}{[x]x \in \Lambda_1(\llcorner)}\ V \qquad \frac{[\Gamma]t \in \Lambda_1(p) \quad [\Delta]u \in \Lambda_1(q)}{[\Gamma,\Delta]t(u) \in \Lambda_1(p(q))}\ A \qquad \frac{[x,\Gamma]t \in \Lambda_1(p)}{[\Gamma]\lambda x.t \in \Lambda_1(\lambda\llcorner.p)}\ L$$

$$\frac{[\Gamma,y,x,\Delta]t \in \Lambda_1(p)}{[\Gamma,x,y,\Delta]t \in \Lambda_1(p)}\ T$$

We write $[\Gamma]t \in \Lambda_1$ to indicate that $[\Gamma]t \in \Lambda_1(p)$ for some p. In general, linear lambda terms should always be considered with free variables indicated, though at times we will leave this implicit. We say that two linear lambda terms $[\Gamma]t, [\Delta]u \in \Lambda_1$ are *α-equivalent* if one can be obtained from the other by renaming of variables (in Γ and Δ, as well as the variables introduced by lambda abstraction within t and u). Linear lambda terms are always considered modulo α-equivalence.

Intuitively, a linear lambda term is *planar* if it is possible to show that it is linear without using the T(ransposition)-rule. Explicitly, this is equivalent to the following definition.

Definition 2.3. Let $[\Gamma]t \in \Lambda_1$ be a linear lambda term. We write $[\Gamma]t \in \Lambda_1^0(p)$ to indicate that t is a *planar lambda term* (with free variables Γ) *decorating* the lambda skeleton p, as defined by the following rules:

$$\frac{}{[x]x \in \Lambda_1^0(\llcorner)}\ V \qquad \frac{[\Gamma]t \in \Lambda_1^0(p) \quad [\Delta]u \in \Lambda_1^0(q)}{[\Gamma,\Delta]t(u) \in \Lambda_1^0(p(q))}\ A \qquad \frac{[x,\Gamma]t \in \Lambda_1^0(p)}{[\Gamma]\lambda x.t \in \Lambda_1^0(\lambda\llcorner.p)}\ L$$

Similarly we write $[\Gamma]t \in \Lambda_1^0$ to indicate that $[\Gamma]t \in \Lambda_1^0(p)$ for some p.

One important observation about planar lambda terms is that they are entirely determined by their lambda skeleton, and conversely, that any lambda skeleton may be decorated by a (necessarily unique) planar lambda term.

Proposition 2.4. *Let* $[\Gamma]t \in \Lambda_1^0(p)$ *and* $[\Delta]u \in \Lambda_1^0(p)$ *be two planar lambda terms decorating the same skeleton* p*. Then* $[\Gamma]t$ *and* $[\Delta]u$ *are* α*-equivalent.*

Proof. Immediate by induction on p. □

$$\frac{}{\langle \llcorner \mid x, \Gamma\rangle \Downarrow \langle x \mid \Gamma\rangle} \qquad \frac{x \text{ fresh} \quad \langle q \mid x, \Gamma\rangle \Downarrow \langle t \mid \Gamma'\rangle}{\langle \lambda\llcorner.q \mid \Gamma\rangle \Downarrow \langle \lambda x.t \mid \Gamma'\rangle} \qquad \frac{\langle p_1 \mid \Gamma\rangle \Downarrow \langle t_1 \mid \Gamma'\rangle \quad \langle p_2 \mid \Gamma'\rangle \Downarrow \langle t_2 \mid \Gamma''\rangle}{\langle p_1(p_2) \mid \Gamma\rangle \Downarrow \langle t_1(t_2) \mid \Gamma''\rangle}$$

FIGURE 1. From lambda skeletons to planar lambda terms

Proposition 2.5. *For any lambda skeleton $p \in \mathcal{S}(i)$, there is a planar lambda term $[x_i, \ldots, x_1]p^\dagger \in \Lambda_1^0(p)$.*

Proof. There is a simple algorithm for computing $p^\dagger$ recursively, by traversing the lambda skeleton p while maintaining a current list of free variables as a stack (initialized to $\Gamma = x_i, \ldots, x_1$, for some i distinct variable names, with x_i at the top):

- (Case $p = \llcorner$): pop a variable name from the top of the stack.
- (Case $p = \lambda\llcorner.q$): generate a fresh variable name not in Γ, push it onto the stack and continue traversing q.
- (Case $p = q(r)$): traverse the application left-to-right, i.e., decorate q (while consuming some variables from the stack) and then decorate r.

In Figure 1 we show this algorithm described in the style of operational semantics, as a relation between a pair of a lambda skeleton and an input stack and a pair of a planar lambda term and an output stack. For example, the skeleton $\lambda\llcorner.\lambda\llcorner.\llcorner(\lambda\llcorner.\llcorner\llcorner) \in \mathcal{S}(0)$ can be decorated by the (closed) planar lambda term $\lambda x.\lambda y.y(\lambda z.zx)$, which is the unique planar decoration of this skeleton up to α-equivalence. □

By consequence of Propositions 2.4 and 2.5, the problem of enumerating planar lambda terms is equivalent to that of enumerating lambda skeletons. From the point of view of lambda calculus, though, it is natural to further restrain the problem by asking that we only count *β-equivalence classes*, which is equivalent (by the normalization theorem for linear lambda calculus) to only counting *β-normal* lambda terms.

Let us recall the following well-known characterization of (β-)normal lambda terms, in mutual induction with so-called "neutral" terms:

- Any variable x is neutral.
- If t is neutral and u is normal then the application $t(u)$ is neutral.
- If t is neutral then t is normal.
- If t is normal then the abstraction $\lambda x.t$ is normal.

These definitions ensure recursively that any term which is neutral or normal cannot contain a β-redex $(\lambda x.t)u$ as a subterm. On the other hand, the standard formulation of neutral and normal terms can also plainly be recast as a property of the underlying lambda skeletons. For reasons which will become apparent in Section 3.2, we refer to the proof that a lambda skeleton is neutral or normal as a "coloring" of that skeleton (and likewise for linear lambda terms, by reference to their underlying skeletons).

Definition 2.6. Letting B and R stand for "blue" and "red", we define two graded sets of lambda skeletons $\mathcal{S}_B(i)$ and $\mathcal{S}_R(i)$, for $i \in \mathbb{N}$, via the following rules:

$$\frac{}{\llcorner \in \mathcal{S}_B(1)}\ v \qquad \frac{p \in \mathcal{S}_B(j) \quad q \in \mathcal{S}_R(k)}{p(q) \in \mathcal{S}_B(j+k)}\ a \qquad \frac{p \in \mathcal{S}_B(i)}{p \in \mathcal{S}_R(i)}\ s \qquad \frac{p \in \mathcal{S}_R(i+1)}{\lambda\llcorner.p \in \mathcal{S}_R(i)}\ \ell$$

Let $p \in \mathcal{S}(i)$ be a lambda skeleton. A *c-coloring* of p, for some $c \in \{B, R\}$, consists of a derivation π of $p \in \mathcal{S}_c(i)$ using the rules v, a, s, and ℓ. We write $\pi : (p \in \mathcal{S}_c(i))$ to indicate that

π is a c-coloring of p. In turn, a c-coloring of a linear lambda term consists of a c-coloring of its lambda skeleton. We say that a linear lambda term is *neutral* if it has a B-coloring, and *normal* if it has a R-coloring.

Proposition 2.7. *A linear lambda term is normal if and only if it has no subterms of the form* $(\lambda x.t)(u)$.

We emphasize that throughout this paper we will be interested in colorings themselves, rather than in the mere fact that a linear lambda term is neutral or normal. In particular, the *size* of a normal or neutral linear lambda term will be defined as a property of its associated coloring.

Definition 2.8. Let $p \in \mathcal{S}(i)$ be a lambda skeleton, and let $\pi : (p \in \mathcal{S}_c(i))$ be a c-coloring of p. The *size* $|\pi|$ of π is defined as the number of uses of the s-rule in π. In turn, the size of a normal or neutral linear lambda term is defined as the size of its associated coloring.

By inspection, any linear lambda term has at most *one* B-coloring or R-coloring, which is why it makes sense to define the size of a neutral or normal linear lambda term as the size of its associated coloring. However, a term could certainly have *both* a B-coloring and a R-coloring – since every neutral lambda term is also normal – and so size is really a property of a given linear lambda term *when viewed as either a neutral term or as a normal term*. Note that this is an instance of the concept of *coherence* of an interpretation of typing judgments (in the sense of Reynolds [27, Ch. 16], viewing the colors B and R as "types"), and we will make further use of this style of definition in Section 4.

Under this definition, for example, the normal linear terms

$$[x]x(\lambda y.y) \qquad [x]\lambda y.yx \qquad [x]\lambda y.xy$$

all have size two, as exhibited by the following R-colorings of their skeletons:

$$\dfrac{\dfrac{}{\llcorner \in \mathcal{S}_B(1)}\,v \quad \dfrac{\dfrac{\dfrac{}{\llcorner \in \mathcal{S}_B(1)}\,v}{\llcorner \in \mathcal{S}_R(1)}\,\boxed{s} \quad \dfrac{}{\lambda\llcorner.\llcorner \in \mathcal{S}_R(0)}\,\ell}{\llcorner(\lambda\llcorner.\llcorner) \in \mathcal{S}_B(1)}\,a}{\llcorner(\lambda\llcorner.\llcorner) \in \mathcal{S}_R(1)}\,\boxed{s} \qquad\qquad \dfrac{\dfrac{}{\llcorner \in \mathcal{S}_B(1)}\,v \quad \dfrac{\dfrac{}{\llcorner \in \mathcal{S}_B(1)}\,v}{\llcorner \in \mathcal{S}_R(1)}\,\boxed{s}}{\llcorner\llcorner \in \mathcal{S}_B(2)}\,a}{\llcorner\llcorner \in \mathcal{S}_R(2)}\,\boxed{s}}{\lambda\llcorner.\llcorner\llcorner \in \mathcal{S}_R(1)}\,\ell$$

Observe, however, that $[x]x(\lambda y.y)$ has size one when viewed as a neutral term rather than as a normal term.

We now turn to the problem of counting neutral and normal planar lambda terms by size, which by Propositions 2.4 and 2.5 is equivalent to counting B-colorings and R-colorings. It is straightforward to go from Definitions 2.6 and 2.8 to the following families of *generating functions* $B_i(z)$ and $R_i(z)$ counting B- and R-colorings by size, where the index i stands for the degree of the underlying lambda skeletons, and the coefficient of z^n in each $B_i(z)$ and $R_i(z)$ counts the total number of B/R-colorings of size n:

$$B_i(z) = [i = 1] + \sum_{j+k=i} B_j(z)R_k(z) \qquad R_i(z) = zB_i(z) + R_{i+1}(z)$$

(Here "$[i = 1]$" denotes the Iverson bracket, 1 if $i = 1$ and 0 otherwise.) Next, we can formally aggregate these families

$$B(z, x) \stackrel{\text{def}}{=} \sum_{i\geq 0} B_i(z)x^i \qquad R(z, x) \stackrel{\text{def}}{=} \sum_{i\geq 0} R_i(z)x^i$$

to define a single pair of generating functions counting colorings along both size and degree. By unfolding definitions, we can then check that $B(z,x)$ and $R(z,x)$ satisfy the following functional equations:

$$B(z,x) = x + B(z,x)R(z,x) \tag{2.1}$$

$$R(z,x) = zB(z,x) + \frac{1}{x}(R(z,x) - R_0(z)) \tag{2.2}$$

Equations of this form can be solved by a technique known as the *quadratic method* [9, p.515]. In particular we can solve for $R_0(z)$, the generating function counting R-colorings of lambda skeletons of degree 0 (hence, closed normal planar lambda terms) by size:

Proposition 2.9. *The generating function $R_0(z)$ satisfies*

$$R_0(z) = -\frac{1}{54z}\left(1 - 18z - (1-12z)^{3/2}\right).$$

Proof. Formula (2.1) becomes

$$B(z,x) = \frac{x}{1 - R(z,x)}$$

and after substituting into (2.2) we derive:

$$\begin{aligned}
R(z,x) &= \frac{zx}{1-R(z,x)} + \frac{1}{x}(R(z,x) - R_0(z)) \\
x(1-R(z,x))R(z,x) &= zx^2 + (1-R(z,x))(R(z,x) - R_0(z)) \\
(x-1)(1-R(z,x))R(z,x) &= zx^2 - (1-R(z,x))R_0(z) \\
((x-1)(1-R(z,x)) - R_0(z))R(z,x) &= zx^2 - R_0(z)
\end{aligned}$$

Then the idea is to define auxiliary functions $F(z,x)$ and $G(z,x)$ by

$$\begin{aligned}
F(z,x) &\stackrel{\text{def}}{=} x - 1 - R_0(z) - 2(x-1)R(z,x) \\
G(z,x) &\stackrel{\text{def}}{=} F(z,x)^2
\end{aligned}$$

and look for a function $X(z)$ such that $F(z,X(z)) = 0$, implying that G has a double root at X. We have chosen $F(z,x)$ so that by the quadratic formula $G(z,x)$ simplifies to

$$G(z,x) = (x - 1 - R_0(z))^2 - 4(x-1)(zx^2 - R_0(z)),$$

and combined with the constraints $G(z,X(z)) = 0$ and $\frac{\partial}{\partial x}G(z,x)|_{x=X(z)} = 0$ we have a system of two equations in two unknowns $X(z)$ and $R_0(z)$. This system of equations can be solved mechanically (for example using Maple),

$$X(z) = \frac{12z + 1 - \sqrt{1-12z}}{18z} \qquad R_0(z) = \frac{(12z-1)X(z) - 8z + 1}{3}$$

and we obtain the stated formula for $R_0(z)$ by algebraic simplification. □

i \ n	0	1	2	3	4	5	6	7	8	9
1	1	1	3	14	83	570	4318	35068	299907	2668994
2	0	1	4	20	120	820	6152	49448	418800	3694740
3	0	0	2	15	105	770	5985	49014	419370	3720420
4	0	0	0	5	56	504	4368	38136	339696	3094896
5	0	0	0	0	14	210	2310	23100	224070	2161236
6	0	0	0	0	0	42	792	10296	116688	1245816

FIGURE 2. The number of neutral planar lambda terms of size n with i free variables.

i \ n	1	2	3	4	5	6	7	8	9	10
0	1	2	9	54	378	2916	24057	208494	1876446	17399772
1	1	2	9	54	378	2916	24057	208494	1876446	17399772
2	0	1	6	40	295	2346	19739	173426	1576539	14730778
3	0	0	2	20	175	1526	13587	123978	1157739	11036038
4	0	0	0	5	70	756	7602	74964	738369	7315618
5	0	0	0	0	14	252	3234	36828	398673	4220722
6	0	0	0	0	0	42	924	13728	174603	2059486

FIGURE 3. The number of normal planar lambda terms of size n with i free variables.

Now, the formula for $R_0(z)$ given in Proposition 2.9 is just one factor of z times the known generating function for counting rooted planar maps by number of edges [9, Proposition VII.11]:

$$-\frac{1}{54z^2}\left(1-18z-(1-12z)^{3/2}\right)$$

Since we also trivially have $R_1(z) = R_0(z)$ (corresponding to the fact that any closed normal lambda term must be a lambda abstraction), we obtain the

Corollary 2.1. *The number of rooted planar maps with n edges is equal to the number of closed normal planar lambda terms (= R-colorings of degree 0) of size $n+1$, and to the number of normal planar lambda terms with one free variable (= R-colorings of degree 1) of size $n+1$.*

From the solution for $R_0(z)$ we can also derive algebraic generating functions for $B(z,x)$ and $R(z,x)$, and use these to compute tables of coefficients, such as the small ones listed in Figures 2 and 3. As a couple of simple observations we note that:

- The series counting neutral planar lambda terms with one free variable (i.e., the coefficients of $B_1(z)$, corresponding to row $i=1$ of Figure 2) also appears in the OEIS as series **A220910**.
- Adding up each column of Figure 2 gives the first row of Figure 3, what can be expressed in generating functions by the equation $R(z,0) = zB(z,1)$. Bijectively, this corresponds to the fact that any closed normal lambda term begins with a series of i lambda abstractions, applied to a neutral term with i free variables.

Finally, although the notion of size given in Definition 2.8 turns out to be a natural one for neutral and normal linear terms, let us point out that it has various equivalent formulations. For example, it is almost identical to counting the total number of variable uses in a term:

Proposition 2.10. *The size of a normal linear lambda term t with R-coloring π is equal to the total number of (V-)leaves in its underlying lambda skeleton, or explicitly $|\pi| = |t|$, where*

$$|\lambda x.t| = |t| \quad \textit{and} \quad |x| = 1 \quad \textit{and} \quad |t(u)| = |t| + |u|.$$

On the other hand the size of a neutral linear lambda term t with B-coloring π is equal to the total number of leaves in its lambda skeleton minus one, i.e., $|\pi| = |t| - 1$.

To prove this, we first recall the standard lambda calculus notion of *head normal form* [4, p.173], which will also be useful in Section 4.

Definition 2.11. Let $\pi : (p \in \mathcal{S}_B(i))$ be a B-coloring of a skeleton p. The *head* of π is defined as the unique occurrence of "$\llcorner$" in p reached by walking up the derivation π from the last rule applied, and always following the left branch of an a-rule until reaching a v-rule. In turn, the *head variable* of a linear lambda term $[\Gamma]t \in \Lambda_1(p)$ with B-coloring $\pi : (p \in \mathcal{S}_B(i))$ is defined as the variable annotating the head of π. (Note that the head variable of a neutral linear term $[\Gamma]t$ necessarily occurs in Γ.) Alternatively, let $\pi : (p \in \mathcal{S}_R(i))$ be a R-coloring of a skeleton p. The *body* of π is defined as the unique subskeleton of p reached by walking up the derivation π from the last rule applied, always moving to the premise of an ℓ-rule until reaching the premise of an s-rule. In turn, the *neutral body* of a linear lambda term $[\Gamma]t \in \Lambda_1(p)$ with R-coloring $\pi : (p \in \mathcal{S}_R(i))$ is defined as the (neutral) linear lambda term annotating the head of π.

For example, y is the head variable of the neutral term $[y, x](y(\lambda z.z))(\lambda w.wx)$, while $[w, x]wx$ is the neutral body of the normal subterm $[x]\lambda w.wx$. To check that this agrees with Definition 2.11, here is the corresponding coloring, where for clarity we have kept all of the variable names decorating the skeleton:

$$\dfrac{\dfrac{\dfrac{}{y \in \mathcal{S}_B(1)}\,v \quad \dfrac{\dfrac{\dfrac{}{z \in \mathcal{S}_B(1)}\,v}{z \in \mathcal{S}_R(1)}\,s}{\lambda z.z \in \mathcal{S}_R(0)}\,\ell}{y(\lambda z.z) \in \mathcal{S}_B(1)}\,a \qquad \dfrac{\dfrac{\dfrac{}{w \in \mathcal{S}_B(1)}\,v \quad \dfrac{\dfrac{}{x \in \mathcal{S}_B(1)}\,v}{x \in \mathcal{S}_R(1)}\,s}{wx \in \mathcal{S}_B(2)}\,a}{wx \in \mathcal{S}_R(2)}\,s}{\lambda w.wx \in \mathcal{S}_R(1)}\,\ell}{(y(\lambda z.z))(\lambda w.wx) \in \mathcal{S}_B(2)}\,a$$

Proof of Proposition 2.10. Let π be the R-coloring of a normal linear lambda term t. There is a one-to-one correspondence between s-nodes in π and variables occurring in t, by walking up from the neutral body of an s-node to the corresponding head variable, and walking back down to the conclusion of the s-node. The same argument works if we begin with a neutral linear term t with B-coloring π, except that the head variable of t itself does not lead back to an s-node. □

2.1. **Related work.** Although lambda calculus is an old subject, its combinatorial aspects have been relatively less studied. In the published literature, Grygiel and Lescanne [12] give recurrence formulas and generating functions for counting pure lambda terms of a given size and number of free variables, as well as for counting normal forms, while David et. al [7] study asymptotic properties of normalization. Most closely related to the present paper are works on the combinatorics of linear lambda calculus (under the alternative name of "BCI" combinatory logic), by Grygiel, Idziak, and Zaionc [11] and by Bodini, Gardy, and Jacquot [5]. Both note a connection between the sequence counting general linear lambda terms (not necessarily β-normal) to series **A062980** of the OEIS, but they do not consider planarity.

3. A graphical language for (neutral and normal) linear lambda terms

Section 4 establishes a bijection between normal planar lambda terms and rooted planar maps, relying on an inductive classification of rooted planar maps due to Tutte. On the other hand, rooted maps also have a very concrete topological interpretation. So, for the purpose of *explaining* the bijection – as well as for better understanding the motivation for studying normal planar lambda terms in the first place – it is helpful to have an analogous graphical representation of linear lambda terms.

The representation we will use is a variation of an old representation sometimes referred to as *lambda-graphs with back-pointers* [2, 20, 32], and which itself can be seen as a variation on linear logic proof-nets [10] adapted to the special case of linear lambda calculus [13]. Other than some superficial syntactic differences, the main refinement we introduce is the addition of a coloring protocol that exactly reflects the restriction of the diagrams to normal and neutral linear lambda terms. Rather than presenting these diagrams as "colored lambda-graphs" or "colored proof-nets", however, one aim of the next section is to explain lambda-graphs within the well-understood framework of *string diagrams*, which were originally introduced by Joyal and Street [16] as a categorical formalization of many different kinds of diagrammatic reasoning (such as Penrose diagrams in physics). As far as we know, this rational reconstruction of lambda-graphs is new, although the ideas we present are quite simple – just enough to motivate the coloring protocol. We will try to keep the exposition relatively elementary, but some background in category theory and lambda calculus may be helpful for reading Section 3.1. On the other hand, the intrepid reader may try skipping straight to Figures 4 and 5 (in Section 3.2) to get a quick feel for the colored diagrams, before heading to Section 4 where we make extensive use of this graphical language.

3.1. **From reflexive objects to lambda-graphs.** Our starting point is the insight, due to Dana Scott [29], that whereas terms of simply-typed lambda calculus can be interpreted as morphisms in arbitrary cartesian closed categories (see, e.g., [17]), terms of pure (or "untyped") lambda calculus can *also* be modelled internally to a cartesian closed category, given an object of that category equipped with a certain special structure (turning it into a so-called *reflexive object*).

Let us recall (for background and details see [19]) that a cartesian closed category can be described as a *closed symmetric monoidal category* in which the monoidal structure is cartesian:

- A *monoidal category* is a category C equipped with a tensor product and unit operation
$$\bullet : C \times C \to C \qquad I : 1 \to C$$
which are associative and unital up to coherent isomorphism.
- It is *closed* if in addition it is equipped with left and right residuation operations
$$\backslash : C^{\mathrm{op}} \times C \to C \qquad / : C \times C^{\mathrm{op}} \to C$$
which are right adjoint to the tensor product in each component:
$$C(y, x \backslash z) \cong C(x \bullet y, z) \cong C(x, z / y)$$
Note that this is equivalent to the existence of a pair of natural transformations
$$C(y, x \backslash z) \xleftarrow{\lambda^x_{y,z}} C(x \bullet y, z) \xrightarrow{\rho^y_{x,z}} C(x, z / y)$$
together with a pair of *evaluation maps*
$$x \bullet (x \backslash z) \xrightarrow{leval_{x,z}} z \xleftarrow{reval_{y,z}} (z / y) \bullet y$$
satisfying equations
$$((\mathrm{id}_x \bullet \lambda^x[f]); leval) = f = ((\rho^y[f] \bullet \mathrm{id}_y); reval)$$
$$g = \lambda^x[(\mathrm{id}_x \bullet g); leval] \qquad \rho^y[(h \bullet \mathrm{id}_y); reval] = h$$
for all morphisms $f : x \bullet y \to z$ and $g : y \to x \backslash z$ and $h : x \to z / y$.
- It is *symmetric* if there is a family of isomorphisms
$$\gamma_{x,y} : x \bullet y \xrightarrow{\sim} y \bullet x$$
which are involutive in the sense that $(\gamma_{x,y}; \gamma_{y,x}) = \mathrm{id}_{x \bullet y}$ for all $x, y \in C$, and which satisfy a few additional, natural equations.
- It is *cartesian* if the tensor product coincides with the categorical product, this meaning that we have a natural isomorphism
$$C(x, y \bullet z) \cong C(x, y) \times C(x, z).$$
Note that this is equivalent to the existence of a family of *duplication* and *erasure* operations
$$\Delta_x : x \to x \bullet x$$
$$e_x : x \to I$$
satisfying certain natural equations.

Any cartesian monoidal category is also symmetric, and the tensor product is usually written $x \bullet y = x \times y$ and called a *categorical product* (or simply a product), while the unit is written $I = 1$. In a closed cartesian monoidal category (more often called a cartesian closed category, or "ccc"), the left and right residuals, which are isomorphic, are usually written $x \backslash y \cong y / x = y^x$ and called *exponential objects*.

Now, Scott defined a *reflexive object* in a ccc C as an object $u \in C$ equipped with a pair of morphisms
$$u \underset{L}{\overset{A}{\rightleftarrows}} u^u$$

such that the $L; A = \mathrm{id}_{u^u}$. The idea is that the two morphisms A and L model the operations of application and lambda abstraction, respectively, while the equation

$$L; A = \mathrm{id}_{u^u} \tag{3.1}$$

models β-conversion (or more precisely β-equivalence). A trivial reflexive object in the cartesian closed category of sets and functions takes u to be the one-element set $1 = \{*\}$, with L and A witnessing the isomorphism $1^1 \cong 1$. In fact, for cardinality reasons, this is the *only* reflexive object in the category of sets and functions. On the other hand, Scott also gave an explicit construction of a non-trivial model of the pure lambda calculus by taking $u = \mathcal{P}(\mathbb{N})$ to be the lattice of subsets of the natural numbers, and u^u to be not the space of *all* functions $\mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$, but rather only those functions preserving directed joins [28]. In terms of the abstract axiomatization introduced in [29], Scott's (earlier) construction could be interpreted as building a reflexive object in the cartesian closed category of *domains and continuous functions*.

It is also possible to consider a dual equation

$$\mathrm{id}_u = A; L \tag{3.2}$$

modeling η-equivalence, which induces an isomorphism $u^u \cong u$, but Scott's definition of reflexive object (and his original model in [28]) only required that u^u be a *retract* of u. A simple but important observation, however, is that the principle of α-equivalence is valid by construction even without either equation (3.1) or (3.2), since the definition itself involves only the two operations A and L, with no mention of formal variables.

The idea here is closely related to a technique sometimes used in programming languages and proof assistants under the heading of *higher-order abstract syntax* (HOAS). Since a reflexive object lives inside a ccc, and since terms of *simply-typed* lambda calculus may be interpreted in any ccc, we can use lambda calculus itself, in addition to the operations A and L, in order to construct morphisms in C denoting pure lambda terms. For example, the closed lambda term

$$\lambda x.xx$$

may be encoded in C as the morphism $\epsilon[\lambda x.xx] : 1 \to u$ defined by

$$\epsilon[\lambda x.xx] = \lambda^u[\Delta; (A \times \mathrm{id}_u); \mathit{reval}]; L \tag{3.3}$$

where we have applied the "currying" transformation λ^u to the morphism

$$u \xrightarrow{\Delta_u} u \times u \xrightarrow{A \times \mathrm{id}_u} u^u \times u \xrightarrow{\mathit{reval}} u$$

to obtain a morphism $1 \to u^u$, and then composed with the operation $L : u^u \to u$. But this can be more slickly written simply as

$$\epsilon[\lambda x.xx] = L(\bar{\lambda} x.A(x)@x)$$

where the "$\bar{\lambda}$" and "@" in the definition of $\epsilon[t]$ correspond to lambda abstraction and application interpreted by appeal to the "meta-level", so to speak – in other words, translated mechanically into the more explicit definition (3.3) by invoking the ccc structure of C. Similarly, the closed term

$$t = (\lambda x.xx)(\lambda y.y)$$

may be encoded in C as the morphism (again of type $1 \to u$)

$$\epsilon[t] \stackrel{\mathrm{def}}{=} A(L(\bar{\lambda} x.A(x)@x))@(L(\bar{\lambda} y.y))$$

and now by purely equational reasoning we can verify, for example, that the morphism encoding t is equal to the morphism encoding $\lambda y.y$:

$$\begin{aligned}
\epsilon[t] &= A(L(\bar{\lambda}x.A(x)@x))@(L(\bar{\lambda}y.y)) && \text{(by definition)}\\
&= (\bar{\lambda}x.A(x)@x)@(L(\bar{\lambda}y.y)) && \text{(by 3.1)}\\
&= A(L(\bar{\lambda}y.y))@(L(\bar{\lambda}y.y)) && \text{(by ccc axioms)}\\
&= (\bar{\lambda}y.y)@(L(\bar{\lambda}y.y)) && \text{(by 3.1)}\\
&= L(\bar{\lambda}y.y) && \text{(by ccc axioms)}\\
&= \epsilon[\lambda y.y] && \text{(by definition)}
\end{aligned}$$

Although it might at first appear circular, this kind of trick is often useful in practice.

Our next step is to observe that Dana Scott's idea also works perfectly well for modelling *linear* lambda calculus in its pure, untyped form, if one simply drops the condition that C be a ccc and replaces it by the weaker condition that C be a closed symmetric monoidal category (smcc). As in a ccc, in a smcc the left and right residuals are isomorphic $x \setminus y \cong y / x$, and they are sometimes denoted collectively by $[x, y]$ (matching the notation for the internal hom in category theory) or by $x \multimap y$ (matching the notation for the implication connective in linear logic). For what comes next, however, it will be important for us to maintain the distinction between the two isomorphic forms of residuals, and moreover to give an explicit name

$$\sigma_{x,y} : x \setminus y \xrightarrow{\sim} y / x$$

for the isomorphism from the left residual to the right residual.

Definition 3.1. A *linear reflexive object* in a smcc C is an object $u \in C$ equipped with a pair of morphisms

$$u \setminus u \xrightarrow{L} u \xrightarrow{A} u / u$$

such that $L; A = \sigma_{u,u}$.

There are certainly some degrees of freedom in this definition that one might consider. For example, one could imagine defining a linear reflexive object as an object equipped with a pair of morphisms

$$u / u \xrightarrow{L'} u \xrightarrow{A'} u \setminus u$$

such that $L'; A' = \sigma^{-1}_{u,u}$, or perhaps as one equipped with a pair

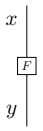
$$u \underset{L''}{\overset{A''}{\rightleftarrows}} u / u$$

such that $L''; A'' = \mathrm{id}_{u/u}$, and so on. We will come back to the difference between these conventions later, but for now we want to take Definition 3.1 as given, and explain how to go from there to a graphical representation of linear lambda terms, by applying the principles of string diagrams more or less mechanically.

We refer to Selinger's survey article [31] for background reading. Briefly, the basic starting point for string diagrams is to dualize the usual object-and-arrow diagrams of category theory, so that objects become (possibly labelled) wires (or "strings"), and arrows

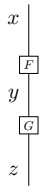
become nodes between wires:

$$x \xrightarrow{F} y \quad \rightsquigarrow$$



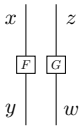
Moreover, composition of morphisms is depicted by connecting diagrams end-to-end,

$$x \xrightarrow{F} y \xrightarrow{G} z \quad \rightsquigarrow$$



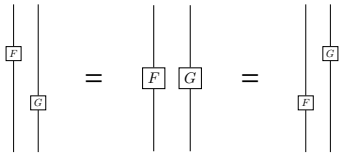
while the tensor product is depicted by laying out diagrams in parallel (with the tensor unit represented by the blank page):

$$x \bullet z \xrightarrow{F \bullet G} y \bullet w \quad \rightsquigarrow$$



$$I \xrightarrow{I} I \quad \rightsquigarrow$$

In general, string diagrams may be treated up to deformation, meaning roughly that it is possible to freely stretch and bend wires and move around nodes, so long as the *interface* of the diagram (i.e., the boundary of input and output wires) remains fixed (for a more precise definition, see [16]). For example, all of the diagrams



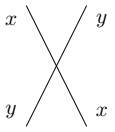
are essentially interchangeable, where the isotopy of diagrams is justified by the equations

$$(F; \mathrm{id}) \bullet (\mathrm{id}; G) = (F \bullet \mathrm{id}); (\mathrm{id} \bullet G) = F \bullet G = (\mathrm{id} \bullet G); (F \bullet \mathrm{id}) = (\mathrm{id}; F) \bullet (G; \mathrm{id})$$

which hold in any monoidal category.

This basic setup may then be developed by supposing that the monoidal category is equipped with additional structure. For example, the symmetry isomorphisms of a symmetric monoidal category are naturally depicted as crossing wires:

$$x \bullet y \xrightarrow{\gamma_{x,y}} y \bullet x \quad \rightsquigarrow$$



Representing the evaluation maps and currying transformations of a smcc is in general a bit more subtle (cf. [3]), but there is a special class of closed symmetric monoidal categories known as *compact closed categories*, which have a particularly simple and elegant graphical language. A symmetric monoidal category is said to be *compact closed* if every object is equipped with left and right *duals*, where a *right dual* of $x \in C$ is an object $x^* \in C$ together with a pair of maps

$$I \xrightarrow{\eta} x \bullet x^* \qquad x^* \bullet x \xrightarrow{\varepsilon} I$$

such that $(\eta \bullet \mathrm{id}_x); (\mathrm{id}_x \bullet \varepsilon) = \mathrm{id}_x$ and $(\mathrm{id}_{x^*} \bullet \eta); (\varepsilon \bullet \mathrm{id}_{x^*}) = \mathrm{id}_{x^*}$, and similarly a *left dual* of $x \in \mathcal{C}$ is an object ${}^*x \in \mathcal{C}$ together with a pair of maps

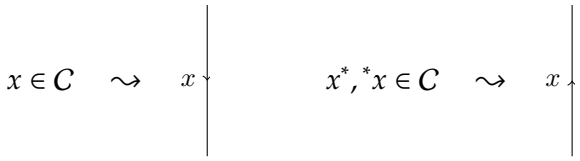
$$I \xrightarrow{\eta'} {}^*x \bullet x \qquad x \bullet {}^*x \xrightarrow{\varepsilon'} I$$

such that $(\eta' \bullet \mathrm{id}_{{}^*x}); (\mathrm{id}_{{}^*x} \bullet \varepsilon') = \mathrm{id}_{{}^*x}$ and $(\mathrm{id}_x \bullet \eta'); (\varepsilon' \bullet \mathrm{id}_x) = \mathrm{id}_x$. Note that any compact closed category is also closed (i.e., has left and right residuals), by defining

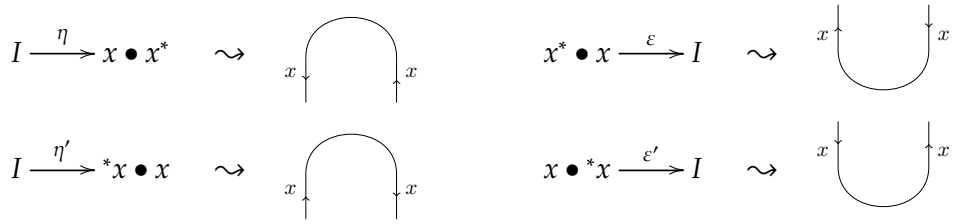
$$x \setminus y \stackrel{\text{def}}{=} {}^*x \bullet y \qquad y / x \stackrel{\text{def}}{=} y \bullet x^* \tag{3.4}$$

and using the maps $\eta^{(\prime)}$ and $\varepsilon^{(\prime)}$ to build the associated currying transformations and evaluation maps. Also note that whenever both left and right duals exist in a symmetric monoidal category they are necessarily isomorphic, and hence the isomorphism ${}^*x \cong x^*$ holds in any compact closed category.

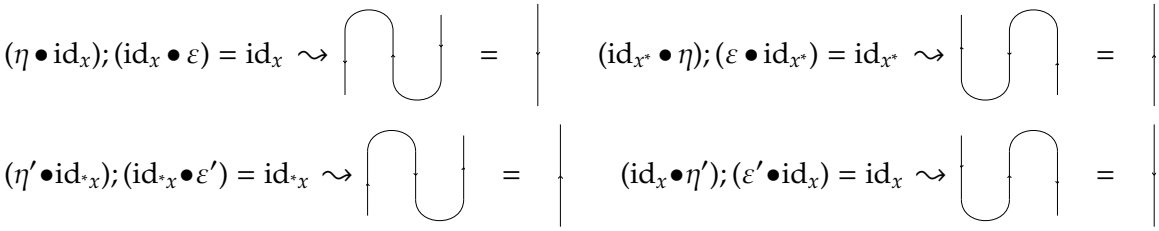
String diagrams for compact closed categories (cf. [31, §4]) are constructed by first assigning *orientations* to the wires to distinguish an object from its duals:



Then, the $\eta^{(\prime)}$ and $\varepsilon^{(\prime)}$ maps are depicted as oriented caps and cups,



while the equations governing them correspond to "straightening" the wires:

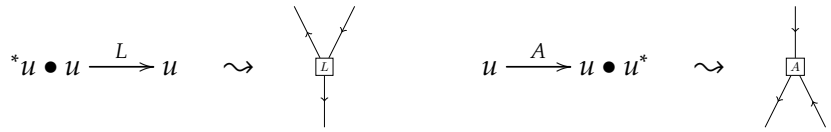


Now, since every compact closed category is also a smcc, we can ask what it means for a compact closed category $\mathcal{C}$ to admit a linear reflexive object. Expanding Definition 3.1 in terms of the canonical description (3.4) of left and right residuals in a compact closed category, a linear reflexive object in $\mathcal{C}$ consists of an object $u \in \mathcal{C}$ equipped with a pair of morphisms

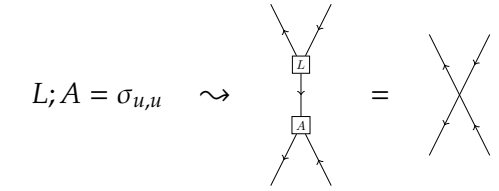
$${}^*u \bullet u \xrightarrow{L} u \xrightarrow{A} u \bullet u^*$$

such that $L; A = \sigma_{u,u}$, where the map $\sigma_{u,u}$ is constructed using the symmetry and the isomorphism ${}^*u \cong u^*$. In turn, following the diagrammatic conventions for compact closed

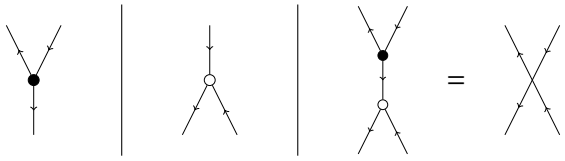
categories, such a structure corresponds to a pair of basic "components" L and A,



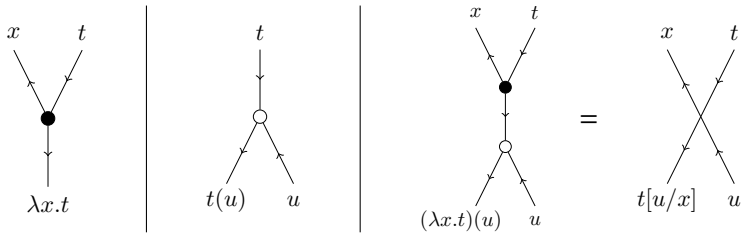
satisfying the following graphical equation:



To see how this plays out in the interpretation of linear lambda terms, let us first take the step of rendering L-nodes by black vertices and A-nodes by white vertices, so as to make the diagrams a bit more evocative:



As suggested by the orientations on the wires, L-nodes and A-nodes can actually be interpreted as *operations on lambda terms*, with certain wires representing inputs and other wires representing outputs. We can visualize this by annotating the wires explicitly with input and output terms:[1]



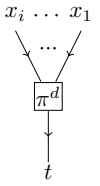
(LR)

Intuitively, an L-node emits a fresh variable x on one wire, then *binds* x in the term t on its incoming wire to produce a term $\lambda x.t$ on its other outgoing wire. Similarly, an A-node takes two lambda terms t and u on its incoming wires, and outputs the application $t(u)$ on its outgoing wire. Moreover, these interpretations are compatible with the graphical interpretation of β-conversion.

Formally, suppose we are given a lambda skeleton $p \in \mathcal{S}(i)$ and a *derivation* π of $[\Gamma]t \in \Lambda_1(p)$, witnessing the fact that t is a linear lambda term decorating p. Then whenever we have a linear reflexive object (u, L, A) in a smcc $\mathcal{C}$, first of all we can define a morphism $[\![\pi]\!]_u : [\![\Gamma]\!]_u \to u$ in $\mathcal{C}$, where $[\![\Gamma]\!]_u \in \mathcal{C}$ is defined inductively by $[\![x]\!]_u = u$, $[\![\Gamma, \Delta]\!]_u =$

[1]These annotations are not to be confused with the convention of labelling wires by objects of the category. Here, every wire represents either the linear reflexive object u or its duals ${}^*u \cong u^*$, and so the orientations suffice as object labels.

$[\![\Gamma]\!]_u \bullet [\![\Delta]\!]_u, [\![\cdot]\!]_u = I$. Moreover, we can define a diagram π^d with i incoming wires and one outgoing wire

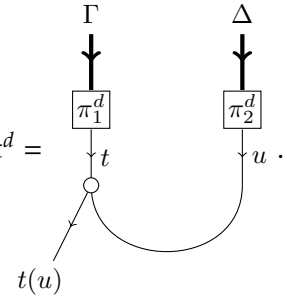


which can be seen as a representation of the image of $[\![\pi]\!]_u$ in the free compact closed category over $\mathcal{C}$. The morphism $[\![\pi]\!]_u$ and diagram π^d are defined by induction on π as follows:

Case $\pi = \dfrac{}{[x]x \in \Lambda_1(\llcorner\lrcorner)}\ V$ **:** Then $[\![\pi]\!]_u = \ u \xrightarrow{\mathrm{id}_u} u$ and $\pi^d = \ \begin{array}{c} x \\ \vdots \\ x \end{array}$. (Draw an oriented wire.)

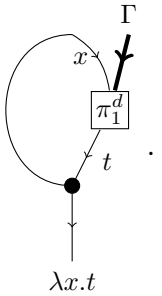
Case $\pi = \dfrac{\overset{\pi_1}{[\Gamma]t \in \Lambda_1(p)} \quad \overset{\pi_2}{[\Delta]u \in \Lambda_1(q)}}{[\Gamma,\Delta]t(u) \in \Lambda_1(p(q))}\ A$ **:** Then

$[\![\pi]\!]_u = \ [\![\Gamma]\!]_u \bullet [\![\Delta]\!]_u \xrightarrow{[\![\pi_1]\!]_u \bullet [\![\pi_2]\!]_u} u \bullet u \xrightarrow{A \bullet \mathrm{id}_u} (u \,/\, u) \bullet u \xrightarrow{\mathit{reval}} u$ and $\pi^d =$



(Connect the outgoing wire of π_2^d and the outgoing wire of π_1^d to an A-node.)

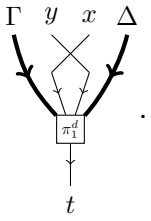
Case $\pi = \dfrac{\overset{\pi_1}{[x,\Gamma]t \in \Lambda_1(p)}}{[\Gamma]\lambda x.t \in \Lambda_1(\lambda\llcorner.p)}\ L$ **:** Then $[\![\pi]\!]_u = \ [\![\Gamma]\!]_u \xrightarrow{\lambda^u[[\![\pi_1]\!]_u]} u \setminus u \xrightarrow{L} u$ and $\pi^d =$



(Connect the outgoing wire and the leftmost incoming wire of π_1^d to an L-node.)

Case $\pi = \dfrac{\overset{\pi_1}{[\Gamma, y, x, \Delta]t \in \Lambda_1(p)}}{[\Gamma, x, y, \Delta]t \in \Lambda_1(p)}\ T$ **:** Then

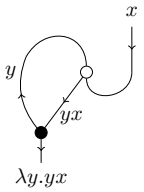
$$[\![\pi]\!]_u = [\![\Gamma]\!]_u \bullet u \bullet u \bullet [\![\Delta]\!]_u \xrightarrow{\mathrm{id}_{[\![\Gamma]\!]_u} \bullet \gamma_{u,u} \bullet \mathrm{id}_{[\![\Delta]\!]_u}} [\![\Gamma]\!]_u \bullet u \bullet u \bullet [\![\Delta]\!]_u \xrightarrow{[\![\pi_1]\!]_u} u \quad \text{and } \pi^d =$$



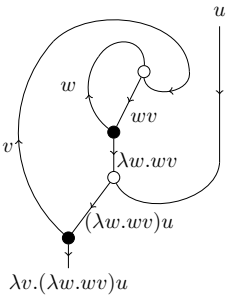
(Crossover the input wires representing x and y.)

Here are some example lambda terms together with their annotated diagrams:

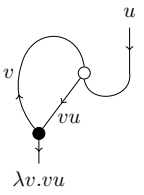
$[x]\lambda y.yx \quad \rightsquigarrow$

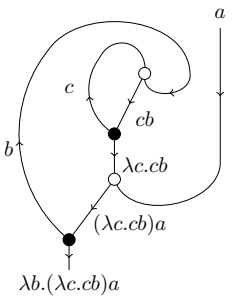


$[u]\lambda v.(\lambda w.wv)u \quad \rightsquigarrow$

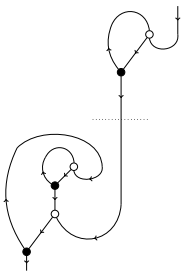


The nice thing is that once we've defined the inductive procedure for translating linear lambda terms (i.e., decorated lambda skeletons) into string diagrams, we can work directly with the diagrams in a much more abstract way. For instance, by the principle of α-conversion, the following are also perfectly legal annotations of the above diagrams:





By looking at the underlying wiring rather than the annotations (which are merely a guide for relating the diagrams to traditional syntax), we can represent lambda terms intrinsically up to renaming of variables. Likewise, another important advantage of string diagrams is that substitution can be represented simply by plugging one diagram into another. For example, the diagram



represents the lambda term $[x]\lambda v.(\lambda w.wv)(\lambda y.yx)$ that results from substitution of the term $[x]\lambda y.yx$ for the variable u in the term $[u]\lambda v.(\lambda w.wv)u$.[2]

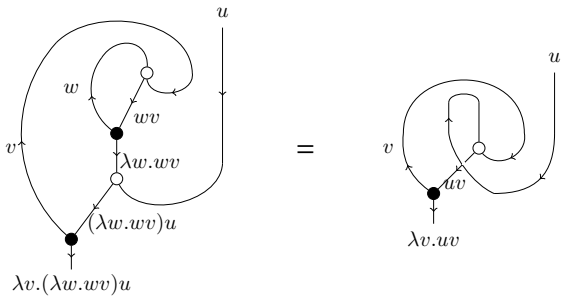
We conclude this section with a few historical and technical remarks:

(1) The graphical language we have derived using the general mechanisms of string diagrams really is not much more than the linear fragment of the language of lambda-graphs (sometimes called "lambda-graphs with back-pointers" [2]). In particular, the idea of representing lambda terms as directed graphs with explicit links from a variable to its binding lambda abstraction may be traced at least as far back as Richard Statman's thesis [32], if not further.[3] Moreover, linear lambda-graphs have a very simple relationship with proof-nets for the implicative fragment of linear logic [13].

(2) However, something to emphasize is that string diagrams are *not* simply special kinds of directed graphs (with two kinds of vertices, and with open edges representing inputs and outputs), but rather they are graphs drawn on the page, and so the order in which wires are positioned around L-nodes and A-nodes matters for determining planarity. Observe that the three example diagrams we showed above are all planar diagrams in the sense that they involve no crossing wires, and indeed the three linear lambda terms

$$[x]\lambda y.yx \qquad [u]\lambda v.(\lambda w.wv)u \qquad [x]\lambda v.(\lambda w.wv)(\lambda y.yx)$$

are all planar in the sense of Definition 2.3. On the other hand, the β-reduction of $[u]\lambda v.(\lambda w.wv)u$ results in a term $[u]\lambda v.uv$ which is not planar in the sense of Definition 2.3, and whose string diagram is not planar in the sense that it contains a crossing:



Now, we need to be a bit careful here: if planarity is really a topological invariant of string diagrams, technically speaking it does not make sense for two diagrams to be equivalent when one is planar and the other is not. But this just reflects the fact that β-reduction is naturally oriented, and our definition of a linear reflexive object (following the pattern of Scott's original definition of a reflexive object in a ccc) does not take that into account. A more "honest" version of Definition 3.1 would take place inside a higher category, so that the rule of β-reduction could be more faithfully described as an oriented cell

$$L; A \Rightarrow \sigma_{u,u}$$

rather than as an equation (cf. [30]). On the other hand, we do not need to pursue this additional level of sophistication here, because in the next section we will describe a

[2]Note that these two advantages – the representation of terms modulo α-equivalence, and the ability to easily express substitution – are well-known arguments for the use of HOAS in proof assistants.

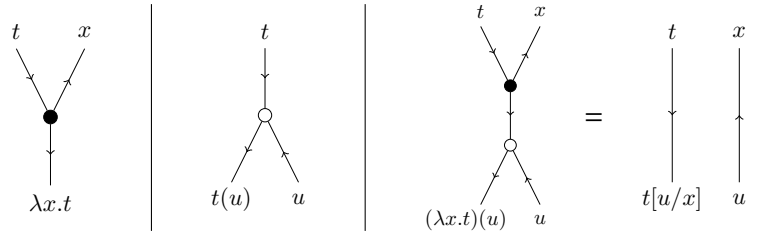
[3]Pierre Lescanne (personal communication) notes that a similar convention appears in the opening chapter of Bourbaki's *Theory of Sets*, as a variable-free syntax for logical formulas involving quantifiers.

simple way of restricting to only the string diagrams which represent β-normal lambda terms, such that this question does not even arise.

(3) Since order matters, some alternative definitions of linear reflexive object would have given rise to a different notion of planarity. For example, if we had asked for a pair of morphisms

$$u \underset{L''}{\overset{A''}{\rightleftarrows}} u \,/\, u$$

such that $L''; A'' = \mathrm{id}_{u/u}$ (or $L''; A'' \Rightarrow \mathrm{id}_{u/u}$), then the various components would be drawn as follows:

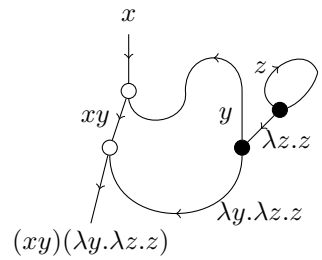


(RL)

In the literature on lambda-graphs, both of the alternative conventions (LR) and (RL) appear. For example, Guerrini [13] displays β-reduction as a crossing, while Mairson [20] displays it as a planar rewriting (and Buliga [6] considers both possibilities). Notably, Abramsky [1] has discussed a notion of planarity that coincides with the fragment of linear lambda terms whose string diagrams are planar by the (RL) convention. For example, the two terms $[x]\lambda y.xy$ and $[x]x(\lambda y.y)$ are planar according to Abramsky's definition ("RL-planar"), whereas the ("LR-planar") term $[x]\lambda y.yx$ is not RL-planar.

There is actually a trivial bijection between LR-planar terms and RL-planar terms, based on the fact that both are fully determined by their lambda skeletons. Indeed, it is possible to adapt the algorithm described in Figure 1 to annotate a lambda skeleton with an RL-planar term, where the only modification needed is that instead of traversing applications left-to-right, they are traversed right-to-left (hence the mnemonics "LR" and "RL"). Therefore, at this level of abstraction, the choice of planarity convention might seem like just a matter of taste. However, we believe the (LR) convention to be more natural when viewing planarity as a *property* of linear lambda terms, rather than as defining an independent "planar lambda calculus". We will provide some evidence for this view in Section 4, by showing that normal LR-planar lambda terms admit a computationally-natural analogue of *Tutte decomposition*. In particular, although the said bijection between LR-planar terms and RL-planar terms preserves the property of being β-normal (so that normal RL-planar terms are also in size-preserving bijection with rooted planar maps), it considerably changes the computational structure of terms.

(4) Finally, let us point out that not every possible string diagram composed out of L-nodes and A-nodes results in a valid linear lambda term. For example, the diagram



only represents a *pseudo* lambda term,[4] which is *ill-scoped* in the sense that the variable y is used before it is bound by λy. This is a phenomenon which is well-known in the literature on lambda-graphs (see, e.g., the *scoped lambda-graphs* of [2]), as well as in the literature on proof-nets (where it leads to so-called *correctness criteria*). In terms of the categorical semantics, this corresponds to the fact that arbitrary diagrams composed of L-nodes and A-nodes can be interpreted as morphisms in any compact closed category containing a linear reflexive object, but not every smcc is compact closed. Again, though, this is perfectly fine for our purposes here, since we will always be able to verify that the diagrams we consider come from the interpretation of a linear lambda term.

3.2. **A coloring protocol for neutral and normal terms.** Now that we have explained the semantic basis of lambda-graphs as string diagrams for linear lambda terms, we will move more quickly in describing how to color these string diagrams to obtain a graphical representation of neutral and normal terms. Our coloring protocol is again derived mechanically from a *refinement* of the definition of a linear reflexive object.

Definition 3.2. A *linear reflexive pair* in a smcc $\mathcal{D}$ is a pair of objects $B, R \in \mathcal{D}$ equipped with a quadruple of morphisms

$$B \setminus R \xrightarrow{\ell} R \underset{s}{\overset{c}{\rightleftarrows}} B \xrightarrow{a} B \,/\, R$$

such that $s; c = \mathrm{id}_B$ and $\ell; c; a = (\mathrm{id}_B \setminus c); \sigma_{b,b}; (\mathrm{id}_B \,/\, c)$.

This is actually a "refinement" in a technical sense: if one ignores the morphism $c : R \to B$ and associated equations (which we shall explain shortly), this is essentially a linear variation of the *refinement type signature* originally presented by Pfenning in [25]. There, he gave an elegant formulation of the standard inductive definition of neutral and normal lambda terms (which we recalled in Section 2), as a refinement of the higher-order abstract syntax representation of lambda terms.[5] Our categorical reformulation is based on a functorial view of type refinement [23], the idea being that one should view a linear reflexive pair in some smcc $\mathcal{D}$ as living over a linear reflexive object in another smcc C, equipped with a (smcc) functor $|-| : \mathcal{D} \to C$ such that

$$|B| = |R| = u \qquad |\ell| = L \qquad |s| = \mathrm{id}_u = |c| \qquad |a| = A.$$

This definition of linear reflexive pair may also be compared to Melliès' definition of *Frobenius pair* [22], which is a refinement of the notion of a Frobenius monoid.

[4] Thanks to Ed Morehouse for this example.

[5] In fact, he considered a slightly more sophisticated, dependently-typed HOAS representation of natural deduction proofs (which are isomorphic to simply-typed lambda terms), and its refinement to represent normal

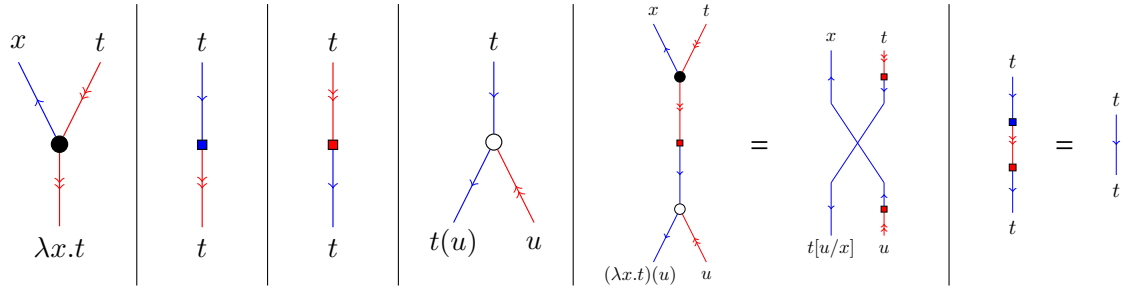
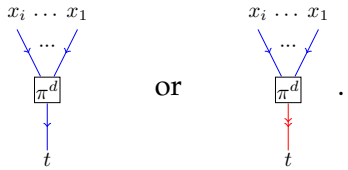


FIGURE 4. Basic components and reduction rules representing a linear reflexive pair, annotated with their corresponding actions on lambda terms.

In Figure 4, we describe the graphical language that results from interpreting a linear reflexive pair in a compact closed category (reading the components left to right as ℓ, s, c, and a, followed by the two equations). The recipe is precisely analogous to the one we detailed in Section 3.1, but we comment on a few points:

(1) The objects B and R are interpreted respectively as blue and red oriented wires. To increase visual contrast and make the diagrams readable without color, we place an extra stroke on red wires.
(2) ℓ-nodes are a colored version of L-nodes, where reading counterclockwise the wires run as follows: outgoing-red, incoming-red, outgoing-blue.
(3) a-nodes are a colored version of A-nodes, where reading counterclockwise the wires run as follows: incoming-blue, outgoing-blue, incoming-red.
(4) The annotations are derived from the forgetful functor $|-| : \mathcal{D} \to \mathcal{C}$ described above. In particular, observe that s-nodes and c-nodes act as identity operations on lambda terms.

Moreover, by a simple extension of the inductive procedure described in Section 3.1, any neutral or normal linear lambda term $[x_i, \ldots, x_1]t$ may be assigned a morphism of the form $[\![\pi]\!] : B \bullet \ldots \bullet B \to B$ or $[\![\pi]\!] : B \bullet \ldots \bullet B \to R$ in any smcc with a linear reflexive pair, as well as a corresponding colored diagram of the form



Diagrams of such neutral or normal terms have the additional property of containing no c-nodes, and in fact, *all* of the diagrams that we consider below have no c-nodes – so it is worth commenting on the presence of the operation $c : R \to B$ and its associated equations in Definition 3.2.

In proof theory and type systems, this technique is actually well-established (cf. [8, 26]): starting from a language restricted to only neutral and normal terms, one can represent arbitrary terms by adding "virtual" coercions from normal to neutral, which can then be eliminated by a process analogous to cut-elimination for sequent calculus. In the presence

and neutral proofs. But the adaptation of the refinement type signature in [25] to the case of pure lambda calculus is straightforward.

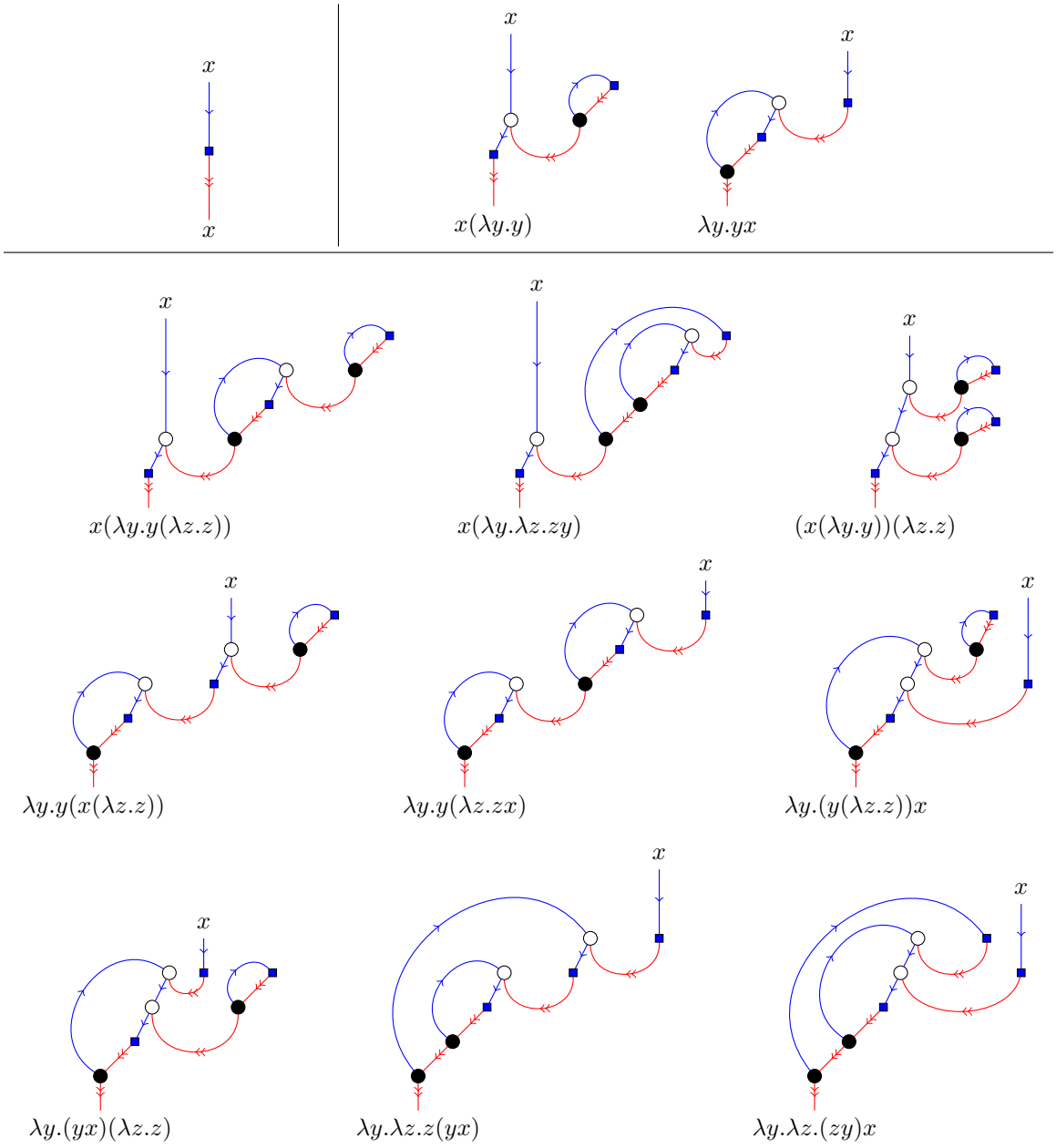


FIGURE 5. String diagrams of the 12 normal planar lambda terms of size ≤ 3 (with one free variable). We annotate the incoming wire with the name of the free variable, and the outgoing wire with the represented lambda term.

of *c*-nodes, any linear lambda term can be represented by many different diagrams, but the normalization theorem for linear lambda calculus implies that these can all be reduced to a unique *c*-node-free one. However, we are not going to study normalization in this paper, and the only reason we give the general definition of a linear reflexive pair and its associated graphical language is because these are natural refinements of the definition and

associated graphical language of a linear reflexive object. In the next section we will consider string diagrams representing normal linear lambda terms, hence which are c-node-free, and moreover we will be mainly interested in the planar case. Figure 5 shows all such (c-node-free, planar) diagrams for the 12 normal planar lambda terms of size ≤ 3, while Appendix A shows all diagrams for the 54 normal planar terms of size = 4.

4. Relating normal planar lambda terms to rooted planar maps via Tutte decomposition

In this section we give our main result, a size-preserving bijection between normal planar lambda terms and rooted planar maps. By "normal planar lambda term", we mean a linear lambda term which

(1) is planar in the sense of Definition 2.3,
(2) is equipped with an R-coloring in the sense of Definition 2.6, and
(3) has one free variable.

Our proof relies on an inductive characterization of rooted planar maps originally described by Tutte [34], so we begin by recalling his analysis in Section 4.1 (for another presentation, see Flajolet and Sedgewick [9, VII.8.2]). The idea will then be to reconstruct Tutte's analysis in the setting of linear lambda calculus, to obtain a "parallel" decomposition of normal planar lambda terms. We show how to do this in Section 4.2, using both traditional lambda calculus notation (following the conventions of Section 2) as well the string diagrams of Section 3. Finally, in Section 4.3 we explain how to combine these parallel analyses to obtain a size-preserving bijection between rooted planar maps and normal planar lambda terms.

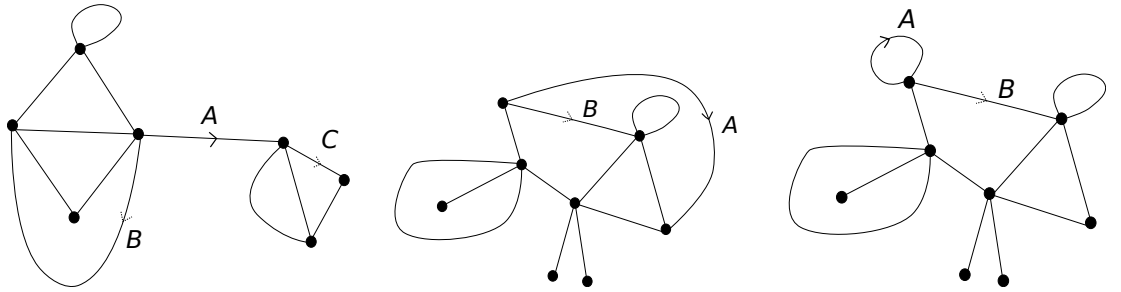
4.1. Tutte decomposition of rooted planar maps.

A *(topological) map* M on a closed, oriented surface S [15, 18] is a partition of S into three finite sets of cells V, E, and F, such that: a *vertex* $v \in V$ is a point of S, an *edge* $e \in E$ is a simple open Jordan arc in S whose extremities are vertices, and a *face* $f \in F$ is a connected component of the complement of $V \cup E$ in S, homeomorphic to an open disk.

An edge equipped with one of two possible orientations is called a *dart*. Each dart d has an *opposite dart* $-d$, corresponding to the same edge with the opposite orientation. The initial vertex (source) of a dart and the face to the left of a dart are both said to be *incident* to that dart. An *isthmus* (resp. *loop*) is an edge whose two orientations are incident to the same face (resp. *vertex*). The *degree* of a vertex or face counts the total number of darts incident to that vertex or face (so that an edge is counted twice in the degree of a face if it is an isthmus, and twice in the degree of a vertex if it is a loop).

A *planar map* is a map on the sphere. Every planar map M has an underlying graph which is a connected planar graph, possibly with loops and multiple edges. A degenerate example of a planar map is the *vertex map* – containing a single vertex, no edges, and a single face – while any other planar map must contain at least one edge. By definition, a *rooting* of a planar map M consists of a choice of a dart, unless M is the vertex map, in which case it is also considered rooted by default. Then, a *rooted planar map* is a planar map equipped with a rooting, treated up to root-preserving homeomorphism. Tutte's analysis begins by noting that any rooted planar map M can be categorized into one of three possible classes:

(1) M is the vertex map.



(a) Map with isthmic root. (b) Map with non-isthmic root. (c) Map with non-isthmic root.

FIGURE 6. Examples of rooted planar maps (root marked A).

(2) M has an *isthmic* root: deleting the root edge separates the underlying graph into two connected components.
(3) M has a *non-isthmic* root: the underlying graph remains connected when the root edge is deleted.

The isthmic and non-isthmic cases are illustrated in Figure 6, with root dart A indicated by an arrow. In these diagrams and more generally, we refer to the face incident to (i.e., to the left of) A as the *outer face* of M (following the convention that we always draw rooted planar maps on the page with the "infinite" face to the left of the root). Tutte's analysis goes on to describe how any map which is not a vertex map may be decomposed in terms of smaller rooted planar maps, by a deterministic procedure:

(a) **Isthmic root.** Let M_1 and M_2 be the two planar maps resulting from deleting the isthmic root A. Each of M_1 and M_2 is either the vertex map, or else may be rooted by walking along the outer face of M and choosing respectively the dart immediately following $-A$ and A (marked B and C in Figure 6a).
(b) **Non-isthmic root.** Let M_1 be the planar map resulting from deleting the non-isthmic root A. Again, if it is the vertex map then we are done, and otherwise M_1 can be rooted by taking the dart (marked B in Figure 6b) immediately following $-A$ when walking along its incident face, unless that dart is $-A$ itself, in which case we take the dart (marked B in Figure 6c) immediately following A along the outer face.

Note that Tutte [34] used slightly different conventions for rooting submaps than what we describe here, but for the purpose of counting rooted planar maps (as was Tutte's original application), the precise convention used does not matter so long as it is deterministic.

Now, let us view (a) as an operation i_d taking a planar map M with an isthmic root as input, and decomposing it into a pair of rooted planar maps M_1 and M_2. There is clearly a reverse operation i_c, which given any pair of rooted planar maps M_1 and M_2 joins them together to create a rooted planar map with an isthmic root (note that this binary operation is "anti-commutative", in the sense that swapping the arguments reverses the orientation of the root dart). We have

$$i_d(i_c(M_1, M_2)) = (M_1, M_2)$$

for any pair of rooted planar maps M_1 and M_2, and conversely

$$M = i_c(i_d(M))$$

for any planar map M with an isthmic root. Moreover, we have that the number of edges in $i_c(M_1, M_2)$ is equal to one plus the sum of the numbers of edges in M_1 and M_2, and that the

degree of the outer face of $i_c(M_1, M_2)$ is equal to two plus the sum of the degrees of the outer faces of M_1 and M_2 (since the outer face is incident to both the root dart and the opposite dart: see Figure 6a, which shows a map with outer face degree nine, constructed from two maps with outer face degrees four and three, respectively).

Similarly, let us view (b) as an operation n_d taking a rooted planar map M with a non-isthmic root as input, and producing a rooted planar map M_1 as output. Going in the other direction, there is a *family* of operations $n_c^{(k)}$, which given any rooted planar map M_1 with outer face of degree $\geq k$, constructs a rooted planar map with a non-isthmic root as follows: starting at the source vertex x of the root dart, walk backwards (i.e. in the opposite direction of the root dart) k darts along the outer face of M_1 until reaching a vertex y, and then add a new edge between x and y with the new root dart oriented from x to y, in such a way that the border of the new root face is composed in this order of the new root dart and the sequence of k darts composing the reverse walk. (For example, the maps in Figures 6b and 6c may be constructed as $n_c^{(8)}(M_1)$ and $n_c^{(11)}(M_1)$, respectively, for the same underlying rooted planar map M_1 of outer face degree 11.)

We have that

$$n_d(n_c^{(k)}(M_1)) = M_1$$

for all rooted planar maps M_1 and k bounded by the degree of the outer face of M_1, and conversely, that there *exists* a k such that

$$M = n_c^{(k)}(n_d(M))$$

for any rooted planar map M with a non-isthmic root. Moreover, we have that the number of edges in $n_c^{(k)}(M_1)$ is equal to one plus the number of edges in M_1, and that the degree of the outer face of $n_c^{(k)}(M_1)$ is $k + 1$.

This combination of observations yields a complete characterization of rooted planar maps, by induction on the number of edges:

Theorem 4.1 (Tutte [34])**.** *Let M be a rooted planar map with $e(M)$ edges and outer face degree $o(M)$. Then exactly one of the following cases must hold:*

(i) *M is the vertex map and $e(M) = o(M) = 0$.*
(ii) *$M = i_c(M_1, M_2)$ for some M_1 and M_2 such that $e(M) = 1 + e(M_1) + e(M_2)$ and $o(M) = 2 + o(M_1) + o(M_2)$.*
(iii) *$M = n_c^{(k)}(M_1)$ for some M_1 and $0 \leq k \leq o(M_1)$ such that $e(M) = 1 + e(M_1)$ and $o(M) = k + 1$.*

4.2. **Decomposition of normal planar lambda terms.** Here and below, we write "NLT" and "NPT" as abbreviations for "normal linear term" and "normal planar term", respectively, it being implicit that we always consider lambda terms with exactly one free variable, unless otherwise stated. (Note that a normal lambda term with an arbitrary number of free variables can always be seen as one with exactly one free variable, by adding or removing leading λs.)

4.2.1. *A trichotomy on normal linear terms.* NLTs (which may or may not be planar) are naturally partitioned into three classes, depending on how the free variable is used.

Definition 4.1. We say that $[x]t$ is *the identity term* if $t = x$, that it is *function-open* if $x(u)$ is a subterm of t (for some u), and that it is *value-open* if $u(x)$ is a subterm of t (for some u).

Proposition 4.2. *Every NLT is either the identity term, function-open, or value-open (mutually exclusively).*

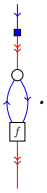
Proof. Immediate by induction, after generalizing the induction hypothesis to consider linear lambda terms with an arbitrary number of free variables. Observe that in pure lambda calculus we have terms such as $[x]\lambda y.x$ which fail to fall into any of these classes, or terms such as $[x]\lambda y.y(xx)$ which classify as both function-open and value-open, but all such counterexamples are ruled out by the requirement that every variable is used exactly once. □

It is informative to restate this classification in terms of linear reflexive pairs and string diagrams.

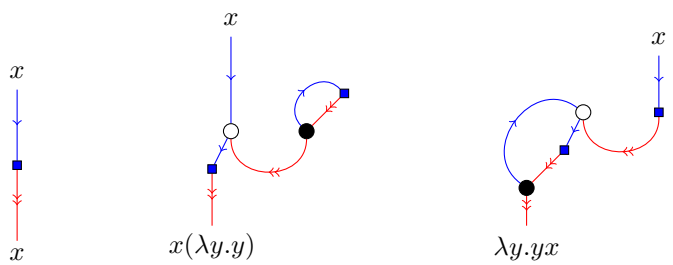
Proposition 4.3. *Let $[x]t$ be a NLT, let $[\![\pi]\!] : B \to R$ be its interpretation in an smcc with a linear reflexive pair, and let π^d be its corresponding string diagram.*

(1) *If $[x]t$ is the identity term then $[\![\pi]\!] = s$ and $\pi^d =$*  .

(2) *If $[x]t$ is function-open then $[\![\pi]\!] = (a; f)$ for some $f : B \,/\, R \to R$ and $\pi^d =$*  .

(3) *If $[x]t$ is value-open then $[\![\pi]\!] = (s; \lambda[a; reval]; f)$ for some $f : B \setminus B \to R$ and $\pi^d =$*  .

Propositions 4.2 and 4.3 apply to arbitrary NLTs, and hence in particular to NPTs. For example, consider once again the diagrams of the first three NPTs:



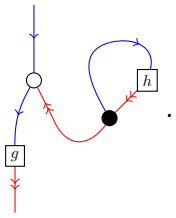
Here we can see that the leftmost diagram corresponds to the identity term, the middle diagram to a function-open term, and the rightmost diagram to a value-open term. Reading

across the rows of the bottom half of Figure 5, we can quickly check that among the nine NPTs of size = 3, the first four are function-open and the next five are value-open.

We now consider how to further decompose the function-open and value-open classes, in the case where the NLT is planar.

4.2.2. *The planar function-open case.*

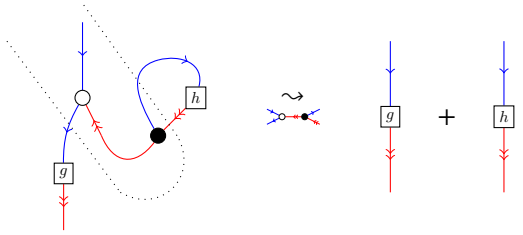
Proposition 4.4. *Let* $[x]t$ *be a NPT, let* $[\![\pi]\!] : B \to R$ *be its interpretation in an smcc with a linear reflexive pair, and let* π^d *be its corresponding string diagram. If* $[x]t$ *is function-open, then*

$[\![\pi]\!] = (a; (\mathrm{id}_{B/R} \bullet (\lambda[h]; \ell)); \mathit{reval}; g)$ *for some* $g : B \to R$ *and* $h : B \to R$, *and* $\pi^d =$ 

.

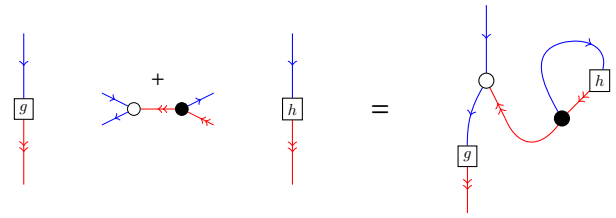
Proof. We get half of the factorization by Proposition 4.3(2). For the second half, we reason that since x is the *only* free variable in t, the constraint of planarity forces the argument of x to be a *closed* NPT, hence of the form $\lambda y.u$ for some u with one free variable y. □

Proposition 4.4 suggests a natural way of performing surgery on the diagram of a function-open NPT, discarding a small piece of π^d to obtain a pair of diagrams with the same interface:

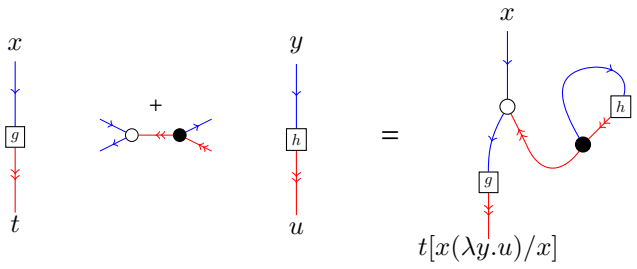


(FO)

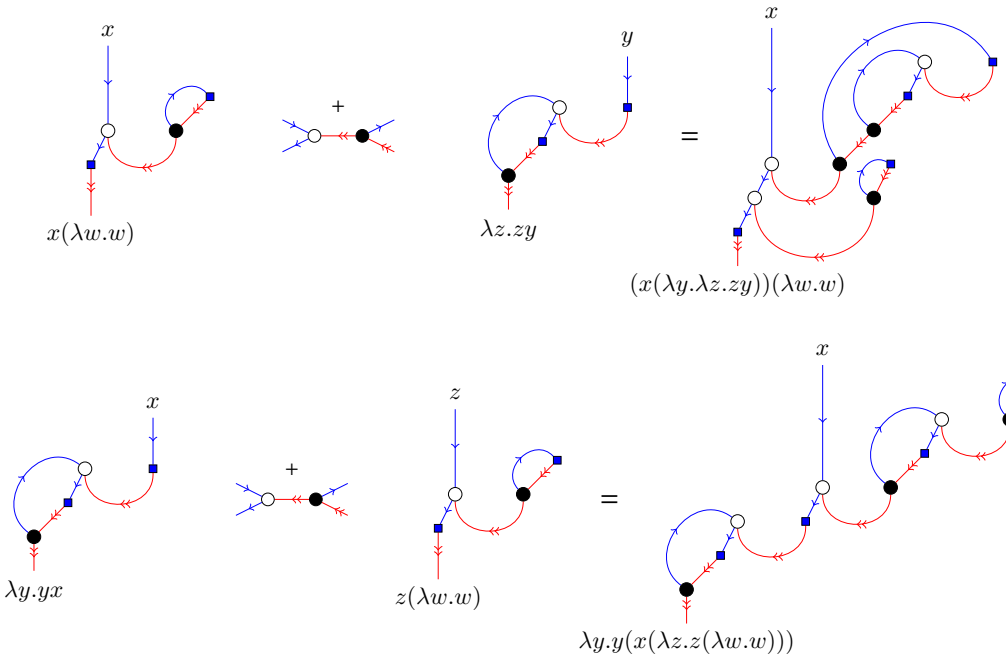
Moreover, this operation is clearly reversible: given any pair of diagrams with one incoming blue wire and one outgoing red wire, we can join them together to obtain a diagram of the original shape:



Finally, if we add annotations to the diagrams,



then we can verify that this operation is easily implemented on NPTs: given a pair of NPTs $[x]t$ and $[y]u$, we produce a new function-open NPT by replacing x with $x(\lambda y.u)$ in t. For example, combining $[x]x(\lambda y.y)$ and $[x]\lambda y.yx$ in either order yields the following two function-open NPTs (after some renaming of variables):



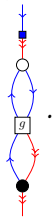
4.2.3. *The planar value-open case.* We begin with an easy observation that holds in the general value-open case.

Proposition 4.5. *If $[x]t$ is a value-open NLT then it must begin with a lambda abstraction, i.e., there exists a normal linear term (with two free variables) $[y_1, x]t'$ such that $t = \lambda y_1.t'$.*

Proof. Let $[\Gamma]u$ be the neutral body of $[x]t$, in the sense of Definition 2.11. By construction, Γ must be of the form $\Gamma = y_i, \ldots, y_1, x$ for some $i \geq 0$. But since $[x]t$ is value-open, x cannot be applied in u, and hence the head variable of u (again in the sense of Definition 2.11) is necessarily distinct from x. This implies that $i > 0$, and the proposition follows. □

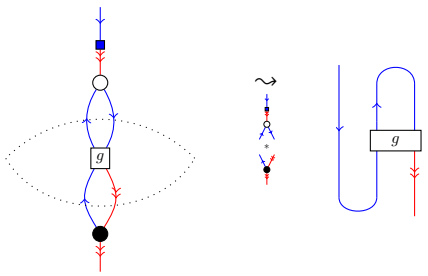
Combining Proposition 4.5 with Proposition 4.3, we obtain the following characterization of value-open NLTs.

Proposition 4.6. *Let $[x]t$ be a NLT, let $[\![\pi]\!] : B \to R$ be its interpretation in an smcc with a linear reflexive pair, and let π^d be its corresponding string diagram. If $[x]t$ is value-open, then*

$[\![\pi]\!] = (s; \lambda[a; reval]; g; \ell)$ *for some* $g : B \setminus B \to B \setminus R$, *and* $\pi^d =$ 

.

Based on this knowledge, here is how we can perform surgery on the diagram of a any value-open NLT to obtain a new diagram with one incoming blue wire and one outgoing red wire:



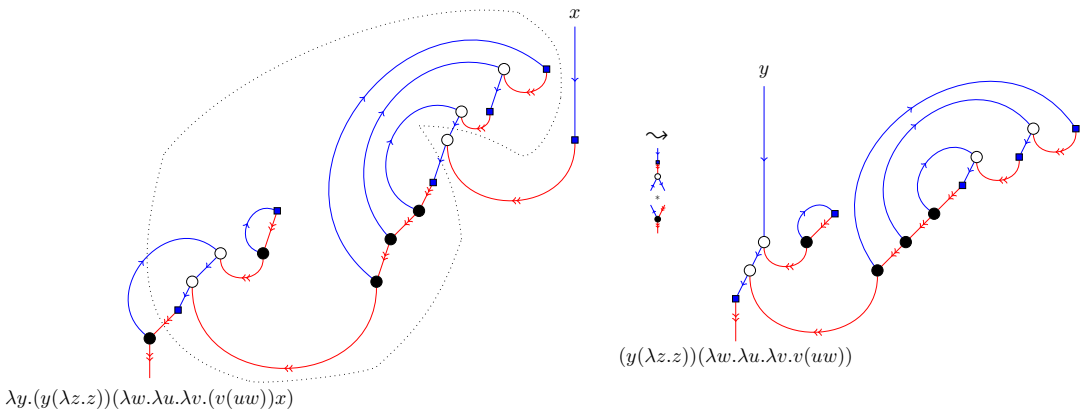
(VO)

In words, the surgery consists of first removing an s-node, an a-node, and an ℓ-node to leave a diagram with four dangling wires, then splicing the two blue wires at the top together, and finally wrapping the blue wire at the bottom back up to the top. In terms of the categorical semantics, surgery (VO) corresponds to extracting the morphism $g : B \setminus B \to B \setminus R$ given by Proposition 4.6, pre-composing it with the currying of the identity morphism on B,

$$I \xrightarrow{\lambda[\mathrm{id}_B]} B \setminus B \xrightarrow{g} B \setminus R$$

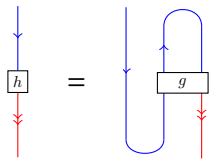
and then uncurrying to obtain a morphism $B \to R$. To borrow terminology from the theory of programming languages, this combined operation can be described as "plugging g with the identity continuation", and we will notate it below by $\langle g \mid \mathrm{id}_B \rangle$. Finally, in terms of Proposition 4.5, the surgery has the effect of simply removing the outermost "λy_1" and the application to x.

Although the (VO) surgery works for any NLT, it is clear that it preserves the planarity of the original diagram. For example, here is a demonstration of surgery on the diagram of a value-open NPT of size 6 (yielding a function-open NPT of size 5):



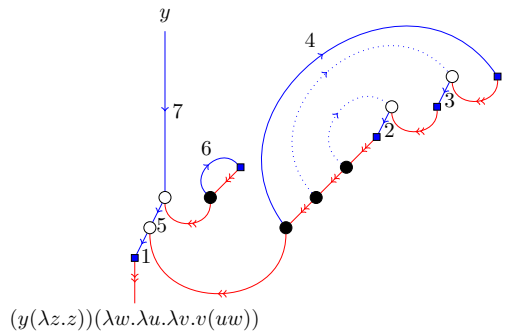
In general, the transformation (VO) is *not* reversible: given a diagram with one incoming blue wire and one outgoing red wire representing a morphism $h : B \to R$, in order to invert the surgery we have to *choose* a particular factorization $h = \langle g \mid \mathrm{id}_B \rangle$ of h as a morphism

$g : B \setminus B \to B \setminus R$ plugged with the identity continuation:



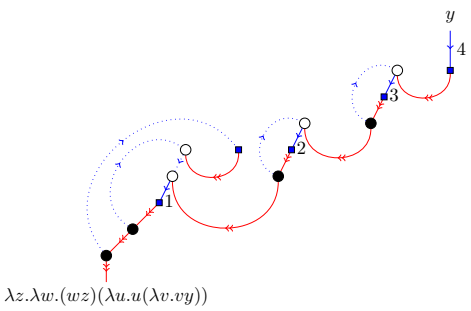
Topologically, such a factorization can be seen as grabbing a blue wire somewhere inside the diagram of h as a "handle", and pulling it to the outside. However, in order to produce a diagram representing a normal planar term, we do not want to consider *arbitrary* factorizations $h = \langle g \mid \mathrm{id}_B \rangle$, but only those factorizations in which the diagram of g is also planar, so that the result of inverting (VO) will be another NPT. A bit of geometric reasoning convinces us that such factorizations should correspond precisely to *blue wires incident to the outer region of the diagram*, where (by analogy to maps) we say that an oriented wire is incident to a region if it has that region to the left, and by "outer region" we mean the open half-plane incident to the incoming and outgoing wires. We shall refer to such blue wires as the *outer neutral handles* of the NPT, deferring a more formal description to Definition 4.7 below.

Consider again the function-open NPT which resulted from value-open surgery above:



Here we have numbered all of the outer neutral handles (while dotting out the remaining blue wires), starting from the bottom of the diagram and walking backwards (i.e., counterclockwise) along the outer region. Each outer neutral handle corresponds to a way of factoring the term by "focusing" on a neutral subterm:

(1) $[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw))$
(2) $[y]\overline{(y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw))}$
(3) $[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.\overline{v(uw)})$
(4) $[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(\underline{uw}))$
(5) $[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw))$
(6) $[y]\overline{(y(\lambda z.z))}(\lambda w.\lambda u.\lambda v.v(uw))$
(7) $[y](\underline{y}(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw))$



(1) $[y]\lambda z.\lambda w.\underline{(wz)}(\lambda u.u(\lambda v.vy))$
(2) $[y]\lambda z.\lambda w.\overline{(wz)(\lambda u.u(\lambda v.vy))}$
(3) $[y]\lambda z.\lambda w.(wz)(\lambda u.\overline{u(\lambda v.vy)})$
(4) $[y]\lambda z.\lambda w.(wz)(\lambda u.u(\lambda v.\overline{vy}))$

$t_1 = [x]\lambda y.\lambda z.\lambda w.((wz)(\lambda u.u(\lambda v.vy)))x$
$t_2 = [x]\lambda y.\lambda z.\lambda w.(wz)(\lambda u.(u(\lambda v.vy))x)$
$t_3 = [x]\lambda y.\lambda z.\lambda w.(wz)(\lambda u.u(\lambda v.(vy)x))$
$t_4 = [x]\lambda y.\lambda z.\lambda w.(wz)(\lambda u.u(\lambda v.v(yx)))$

FIGURE 7. Diagram of a NPT with outer neutral handles indicated and numbered. On the right, we show the associated factorizations of the NPT into a neutral subterm and its surrounding context, as well as the corresponding value-open NPTs which result from inverting the transformation (VO).

Performing an inverse (VO) operation while focused on any of these subterms yields a different value-open NPT:

$t_1 = [x]\lambda y.((y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw)))x$
$t_2 = [x]\lambda y.(y(\lambda z.z))(\lambda w.\lambda u.\lambda v.(v(uw))x)$
$t_3 = [x]\lambda y.(y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v((uw)x))$
$t_4 = [x]\lambda y.(y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(u(wx)))$
$t_5 = [x]\lambda y.((y(\lambda z.z))x)(\lambda w.\lambda u.\lambda v.v(uw))$
$t_6 = [x]\lambda y.(y(\lambda z.zx))(\lambda w.\lambda u.\lambda v.v(uw))$
$t_7 = [x]\lambda y.((yx)(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(uw))$

In turn, performing (VO) on any of the t_k (i.e., erasing x and removing the leading λy) yields back the original function-open NPT. Observe that certain ways of focusing on a neutral subterm are excluded because they do not correspond to *outer* neutral handles. For example, attempting to perform an inverse (VO) operation starting from the factorization

$$[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(\underline{u}w))$$

would indeed result in a value-open NLT,

$$[y](y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v(\underline{u}w)) \quad \rightsquigarrow \quad [x]\lambda y.(y(\lambda z.z))(\lambda w.\lambda u.\lambda v.v((ux)w))$$

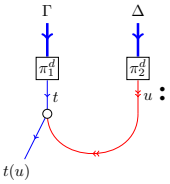
but one which is not planar. In Figure 7 we give another example of a NPT with outer neutral handles indicated, as well as the associated value-open NPTs that arise by inverting (VO).

With that by way of geometric intuition, we can now give a formal specification of the outer neutral handles of a NPT, defining these by induction for any neutral or normal planar term with any number of free variables.

Definition 4.7. Let $[\Gamma]t$ be a neutral or normal planar lambda term. A *neutral handle* of $[\Gamma]t$ is a factorization of t into a neutral subterm and its surrounding context. The set $O(\pi)$ of *outer neutral handles* is defined by induction on the coloring π of $[\Gamma]t$, as follows:

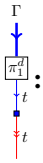
Case $\pi^d =$

: Then $O(\pi) = \{\, \underline{x} \,\}$.

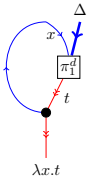
Case $\pi^d =$ 

:

$$\text{Then } O(\pi) = \begin{cases} \{\, \underline{t(u)} \,\} \cup \{\, t(H) \mid H \in O(\pi_2) \,\} \cup \{\, H(u) \mid H \in O(\pi_1) \,\} & \text{if } |\Delta| = 0 \\ \{\, \underline{t(u)} \,\} \cup \{\, t(H) \mid H \in O(\pi_2) \,\} & \text{if } |\Delta| > 0 \end{cases} .$$

Case $\pi^d =$ 

: Then $O(\pi) = \{\, H \mid H \in O(\pi_1) \,\}$.

Case $\pi^d =$ 

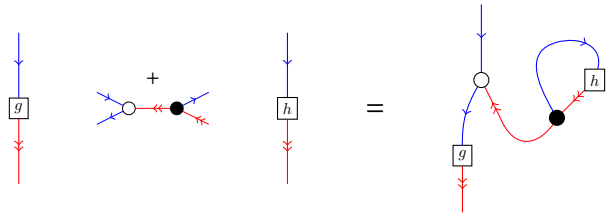
: Then $O(\pi) = \{\, \lambda x.H \mid H \in O(\pi_1) \,\}$.

4.3. **The size-preserving bijection.** Let us write $f_c(t_1, t_2)$ for the binary operation taking a pair of NPTs t_1 and t_2 and joining them together to form a function-open NPT by the procedure described in Section 4.2.2. Similarly, we write $v_c^{(k)}(t_1)$ for the operation taking a NPT t_1 with $\geq k$ outer neutral handles and factoring it along the kth to produce a value-open NPT by the procedure described in Section 4.2.3. We now establish a lambda calculus analogue of Theorem 4.1:

Theorem 4.2. *Let $[x]t$ be a NPT with R-coloring π. Then exactly one of the following cases must hold:*

(i) *$[x]t$ is the identity term and $|\pi| = |O(\pi)| = 1$.*
(ii) *$[x]t = f_c([x_1]t_1, [x_2]t_2)$ for some $[x_1]t_1$ and $[x_2]t_2$ (with R-colorings π_1 and π_2) such that $|\pi| = |\pi_1| + |\pi_2|$ and $|O(\pi)| = 1 + |O(\pi_1)| + |O(\pi_2)|$.*
(iii) *$[x]t = v_c^{(k)}([x_1]t_1)$ for some t_1 (with R-coloring π_1) and $1 \leq k \leq |O(\pi_1)|$ such that $|\pi| = 1 + |\pi_1|$ and $|O(\pi)| = k + 1$.*

Proof. Following Proposition 4.2 and the discussions in Sections 4.2.2 and 4.2.3, what is left to verify is that the operations f_c and $v_c^{(k)}$ have the right effect on the numbers of s-nodes and outer neutral handles. Consider the f_c operation yielding a function-open NPT, as expressed in string diagrams:



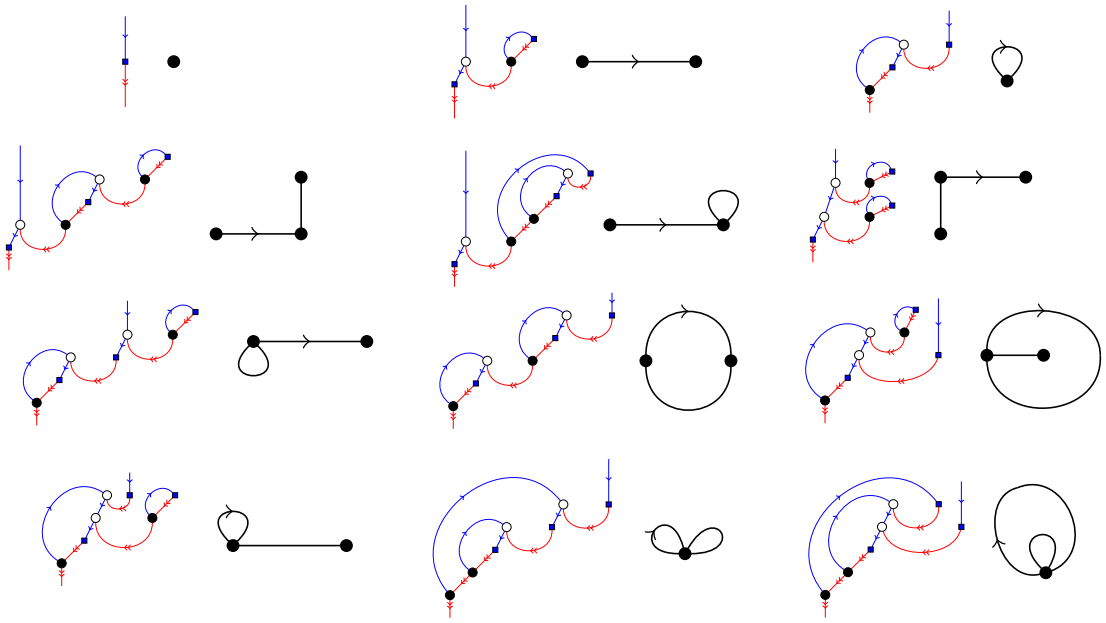
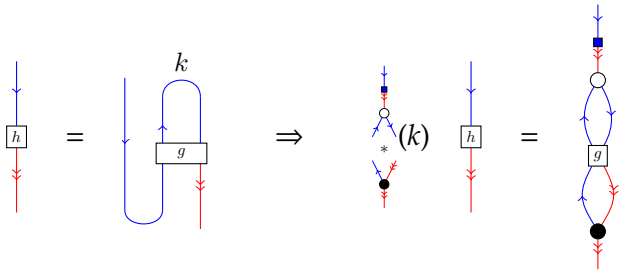


FIGURE 8. Correspondence between the NPTs of size ≤ 3 and rooted planar maps with ≤ 2 edges.

By inspection, the resulting diagram includes all and only the s-nodes coming from the two input diagrams, and all of the outer neutral handles plus one additional one leading into the a-node. Likewise, consider the $v_c^{(k)}$ operation yielding a value-open NPT:



By inspection, the resulting diagram has one additional s-node, and exactly $k+1$ outer neutral handles, corresponding to the first k outer neutral handles of the input diagram plus one additional one leading into the s-node. □

Theorem 4.3. *There is a one-to-one correspondence between rooted planar maps with n edges and outer face degree d, and NPTs with $n+1$ s-nodes and $d+1$ outer neutral handles.*

Proof. By playing Theorems 4.1 nd 4.2 in parallel, to decompose a rooted planar map/NPT and then recompose it as the corresponding NPT/rooted planar map. (This uses an induction on the number of edges of a rooted planar map, and on the size of an NPT.) Note that the off-by-one offset between number of edges/outer face degree and size/number of outer neutral handles means that we need to apply the arithmetic identities $(n+1)+(n'+1) = 1+(n+n'+1)$ and $1+(d+1)+(d'+1) = (2+d+d')+1$ in the isthmic/function-open case. □

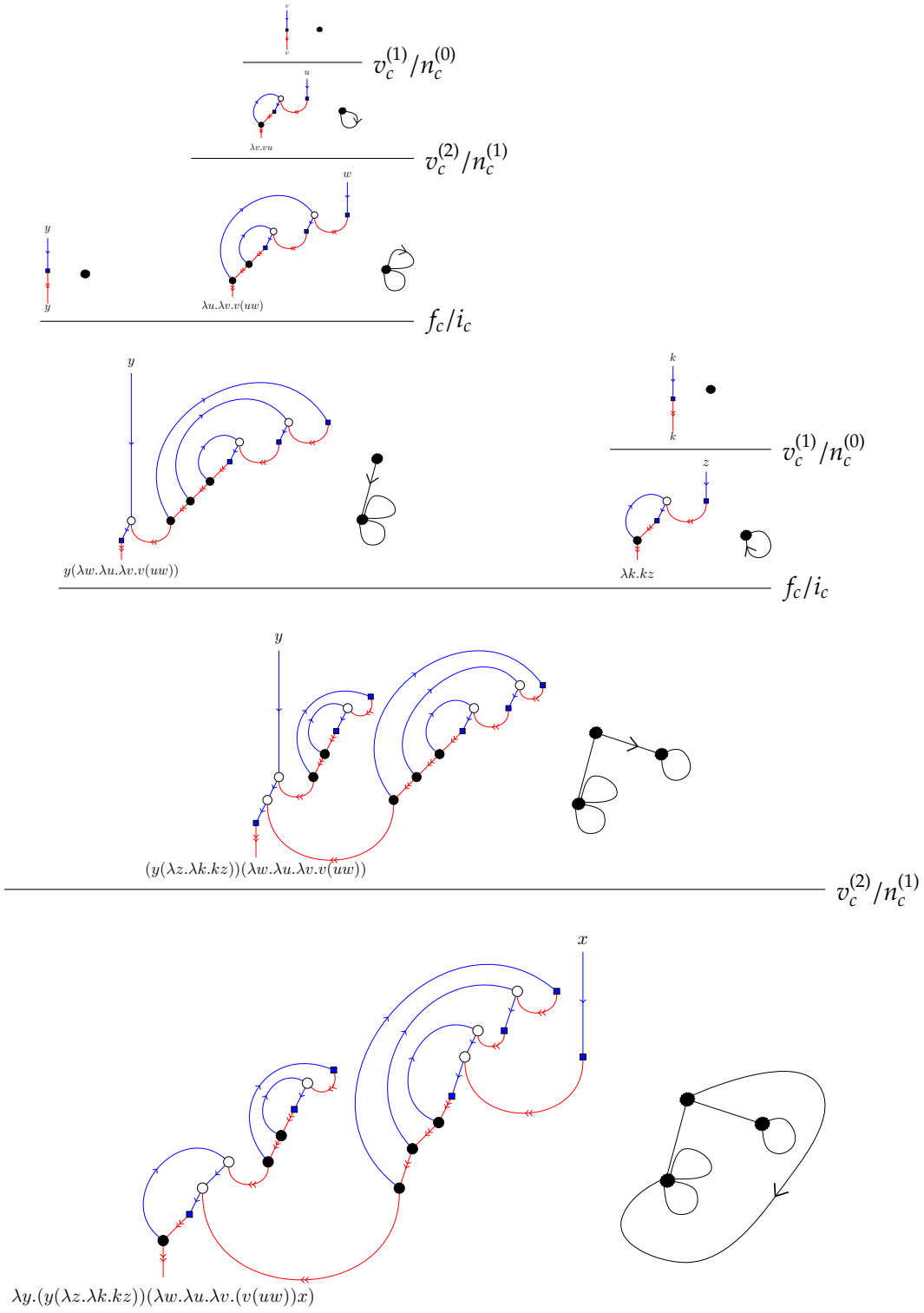


FIGURE 9. Full decomposition of a normal planar lambda term with seven s-nodes and three outer neutral handles, in parallel with the corresponding rooted planar map with six edges and outer face degree two.

In Figure 8, we show the result of applying this bijection to all NPTs of size at most three, while in Figure 9, we give an illustration of the proof of Theorem 4.3 in action, animating the full decomposition of a particular NPT (with 7 s-nodes and 3 outer neutral handles) in parallel with the decomposition of the corresponding rooted planar map (# edges = 6, outer face degree = 2).

Corollary 4.4. *The following families of objects are all in size-preserving bijection:*

- *rooted planar maps*
- *normal planar lambda terms*
- *R-colorings of lambda skeletons*

Acknowledgments

We thank Alexis Saurin, Paul-André Melliès, Maciej Dołęga, and Beniamino Accattoli for discussions and pointers to related work. We also thank Pierre Lescanne for an invitation to speak about this work at the 8th Workshop on Computational Logic and Applications at Lyon in March 2015. The string diagrams in the paper were manipulated with the help of the TikZiT diagram editor.

References

[1] Samson Abramsky. Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. In *Mathematics of Quantum Computing and Technology* (eds. G. Chen, L. Kauffman and S. Lomonaco), 415–458, 2008.

[2] Zena M. Ariola and Stefan Blom. Cyclic Lambda Calculi. In *Proceedings of the Third International Symposium on Theoretical Aspects of Computer Science*, 77–106, Sendai, Japan, 1997.

[3] John Baez and Mike Stay. Physics, Topology, Logic and Computation: A Rosetta Stone. In *New Structures for Physics* (ed. Bob Coecke), Springer Lecture Notes in Physics 813, 95–174, 2011.

[4] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic 103, second, revised edition, North-Holland, Amsterdam, 1984.

[5] O. Bodini, D. Gardy, and A. Jacquot. Asymptotics and random sampling for BCI and BCK lambda terms. *Theoretical Computer Science*, 502:227–238, 2013.

[6] Marius Buliga. Graphic Lambda Calculus. *Complex Systems*, 22(4):311–360, 2013.

[7] René David, Katarzyna Grygiel, Jakub Kozik, Christophe Raffalli, Guillaume Theyssier, and Marek Zaionc. Asymptotically almost all lambda terms are strongly normalizing. *Logical Methods in Computer Science* 9(1:02):1–30, 2013.

[8] Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, 198–208, Portland, Oregon, USA, 2000.

[9] Philippe Flajolet and Robert Sedgewick. Analytic Combinatorics. Cambridge University Press, 2009.

[10] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[11] Katarzyna Grygiel, Pawel M. Idziak, Marek Zaionc. How big is BCI fragment of BCK logic. *Journal of Logic and Computation* 23(3):673–691, 2013.

[12] Katarzyna Grygiel and Pierre Lescanne. Counting and generating lambda terms. *Journal of Functional Programming*, 23(5):594–628, 2013.

[13] Stefano Guerrini. Proof nets and the lambda-calculus. In *Linear Logic in Computer Science* (eds. Thomas Ehrhard, Paul Ruet, Jean-Yves Girard and Philip Scott), 65–118, Cambridge University Press, 2004.

[14] Martin Hyland. Classical lambda calculus in modern dress. To appear in *Mathematical Structures in Computer Science*. arXiv:1211.5762

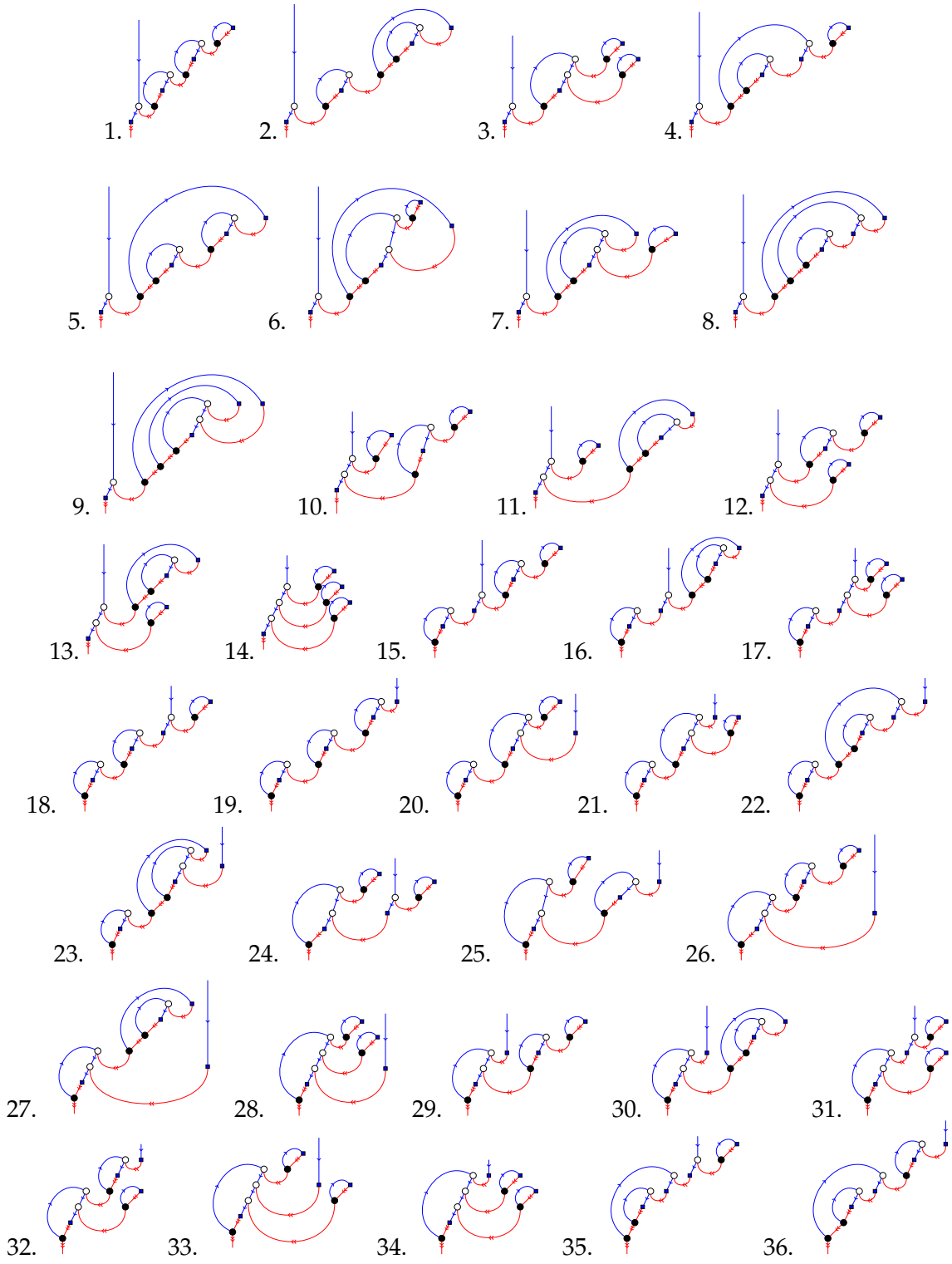
[15] Gareth A. Jones and David Singerman. Theory of maps on orientable surfaces. *Proceedings of the London Mathematical Society*, 37:273–307, 1978.

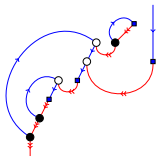
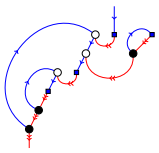
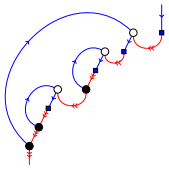
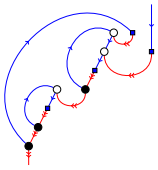
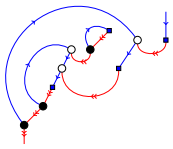
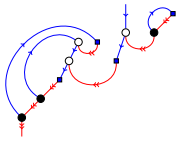
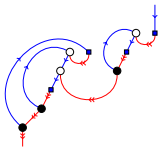
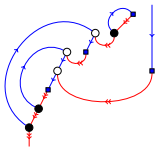
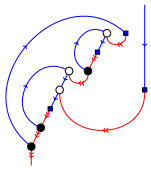
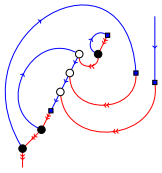
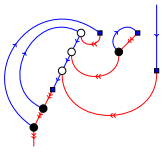
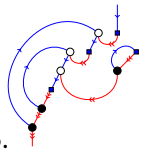
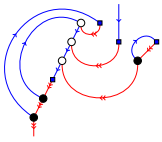
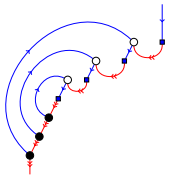
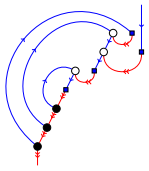
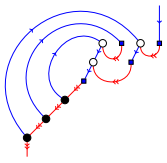
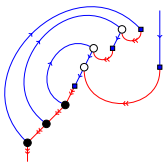
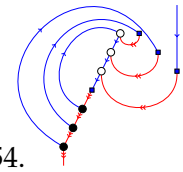
[16] André Joyal and Ross Street. The geometry of tensor calculus I. *Advances in Mathematics*, 102:20–78, 1993.

[17] Joachim Lambek and Philip Scott. *Introduction to Higher-order Categorical Logic.* Cambridge University Press, 1986.
[18] Sergei K. Lando and Alexander K. Zvonkin. *Graphs on Surfaces and Their Applications*, Encyclopaedia of Mathematical Sciences 141, Springer-Verlag, 2004.
[19] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.
[20] 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.
[21] Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming* 14(6), 623–633, 2004.
[22] Paul-André Melliès. Dialogue categories and Frobenius monoids. In *Computation, Logic, Games, and Quantum Foundations: Essays Dedicated to Samson Abramsky on the Occasion of his 60th Birthday* (eds. Bob Coecke, Luke Ong, Prakash Panangaden), Springer LNCS 7860, 197–224, 2013.
[23] Paul-André Melliès and Noam Zeilberger. Functors are Type Refinement Systems. In *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming*, 3–16, Mumbai, India, 2015.
[24] OEIS Foundation Inc. (2011), *The On-Line Encyclopedia of Integer Sequences*, published electronically at http://oeis.org.
[25] Frank Pfenning. Refinement Types for Logical Frameworks. In *Informal Proceedings of the Workshop on Types for Proofs and Programs* (ed. Herman Geuvers), 285–299, Nijmegen, The Netherlands, May 1993.
[26] Jeff Polakow and Frank Pfenning. Relating Natural Deduction and Sequent Calculus for Intuitionistic Non-Commutative Linear Logic. In *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, ENTCS 20, 1999.
[27] John C. Reynolds. *Theories of Programming Languages.* Cambridge University Press, 1998.
[28] Dana S. Scott. Data types as lattices. *SIAM Journal on Computing*, 5(3):522–587, September 1976.
[29] Dana S. Scott. Relating theories of the λ-calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism* (eds. Hindley and Seldin), Academic Press, 403–450, 1980.
[30] R. A. G. Seely. Modelling Computations: A 2-Categorical Framework. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science*, 65–71, Ithaca, NY, USA, 1987.
[31] 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.
[32] Richard Statman. *Structural Complexity of Proofs*, Ph.D. thesis, Stanford University, 1974.
[33] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.
[34] W. T. Tutte. On the enumeration of planar maps. *Bulletin of the American Mathematical Society*, 74:64–74, 1968.

Appendix A. All normal planar lambda terms of size four

Unannotated string diagrams:



37. 
38. 
39. 
40. 
41. 
42. 
43. 
44. 
45. 
46. 
47. 
48. 
49. 
50. 
51. 
52. 
53. 
54. 

Corresponding normal planar lambda terms (with one free variable x):

(1) $x(\lambda y.y(\lambda z.z(\lambda w.w)))$
(2) $x(\lambda y.y(\lambda z.\lambda w.wz))$
(3) $x(\lambda y.(y(\lambda z.z))(\lambda w.w))$
(4) $x(\lambda y.\lambda z.z(y(\lambda w.w)))$
(5) $x(\lambda y.\lambda z.z(\lambda w.wy))$
(6) $x(\lambda y.\lambda z.(z(\lambda w.w))y)$
(7) $x(\lambda y.\lambda z.(zy)(\lambda w.w))$
(8) $x(\lambda y.\lambda z.\lambda w.w(zy))$
(9) $x(\lambda y.\lambda z.\lambda w.(wz)y)$
(10) $(x(\lambda y.y))(\lambda z.z(\lambda w.w))$
(11) $(x(\lambda y.y))(\lambda z.\lambda w.wz)$
(12) $(x(\lambda y.y(\lambda z.z)))(\lambda w.w)$
(13) $(x(\lambda y.\lambda z.zy))(\lambda w.w)$
(14) $((x(\lambda y.y))(\lambda z.z))(\lambda w.w)$
(15) $\lambda y.y(x(\lambda z.z(\lambda w.w)))$
(16) $\lambda y.y(x(\lambda z.\lambda w.wz))$
(17) $\lambda y.y((x(\lambda z.z))(\lambda w.w))$
(18) $\lambda y.y(\lambda z.z(x(\lambda w.w)))$
(19) $\lambda y.y(\lambda z.z(\lambda w.wx))$
(20) $\lambda y.y(\lambda z.(z(\lambda w.w))x)$
(21) $\lambda y.y(\lambda z.(zx)(\lambda w.w))$
(22) $\lambda y.y(\lambda z.\lambda w.w(zx))$
(23) $\lambda y.y(\lambda z.\lambda w.(wz)x)$
(24) $\lambda y.(y(\lambda z.z))(x(\lambda w.w))$
(25) $\lambda y.(y(\lambda z.z))(\lambda w.wx)$
(26) $\lambda y.(y(\lambda z.z(\lambda w.w)))x$
(27) $\lambda y.(y(\lambda z.\lambda w.wz))x$
(28) $\lambda y.((y(\lambda z.z))(\lambda w.w))x$
(29) $\lambda y.(yx)(\lambda z.z(\lambda w.w))$
(30) $\lambda y.(yx)(\lambda z.\lambda w.wz)$
(31) $\lambda y.(y(x(\lambda z.z)))(\lambda w.w)$
(32) $\lambda y.(y(\lambda z.zx))(\lambda w.w)$
(33) $\lambda y.((y(\lambda z.z))x)(\lambda w.w)$
(34) $\lambda y.((yx)(\lambda z.z))(\lambda w.w)$
(35) $\lambda y.\lambda z.z(y(x(\lambda w.w)))$
(36) $\lambda y.\lambda z.z(y(\lambda w.wx))$
(37) $\lambda y.\lambda z.z((y(\lambda w.w))x)$
(38) $\lambda y.\lambda z.z((yx)(\lambda w.w))$
(39) $\lambda y.\lambda z.z(\lambda w.w(yx))$
(40) $\lambda y.\lambda z.z(\lambda w.(wy)x)$
(41) $\lambda y.\lambda z.(z(\lambda w.w))(yx)$
(42) $\lambda y.\lambda z.(zy)(x(\lambda w.w))$
(43) $\lambda y.\lambda z.(zy)(\lambda w.wx)$
(44) $\lambda y.\lambda z.(z(y(\lambda w.w)))x$
(45) $\lambda y.\lambda z.(z(\lambda w.wy))x$
(46) $\lambda y.\lambda z.(z(\lambda w.w)y)x$
(47) $\lambda y.\lambda z.((zy)(\lambda w.w))x$
(48) $\lambda y.\lambda z.(z(yx))(\lambda w.w)$
(49) $\lambda y.\lambda z.((zy)x)(\lambda w.w)$
(50) $\lambda y.\lambda z.\lambda w.w(z(yx))$
(51) $\lambda y.\lambda z.\lambda w.w((zy)x)$
(52) $\lambda y.\lambda z.\lambda w.(wz)(yx)$
(53) $\lambda y.\lambda z.\lambda w.(w(zy))x$
(54) $\lambda y.\lambda z.\lambda w.((wz)y)x$