

Polynomial functors and opetopes

JOACHIM KOCK, ANDRÉ JOYAL, MICHAEL BATANIN, and JEAN-FRANÇOIS MASCARI

Abstract

We give an elementary and direct combinatorial definition of opetopes in terms of trees, well-suited for graphical manipulation and explicit computation. To relate our definition to the classical definition, we recast the Baez-Dolan slice construction for operads in terms of polynomial monads: our opetopes appear naturally as types for polynomial monads obtained by iterating the Baez-Dolan construction, starting with the trivial monad. We show that our notion of opetope agrees with Leinster's. Next we observe a suspension operation for opetopes, and define a notion of stable opetopes. Stable opetopes form a least fixpoint for the Baez-Dolan construction. A final section is devoted to example computations, and indicates also how the calculus of opetopes is well-suited for machine implementation.

Introduction

Among a dozen or so existing definitions of weak higher categories, the opetopic approach is one of the most intriguing, since it is based on a collection of 'shapes' that had not previously been studied: the opetopes. Opetopes are combinatorial structures parametrising higher-dimensional many-in/one-out operations, and can be seen as higher-dimensional generalisations of trees. They are important combinatorial structures on their own, 'as pervasive in higher-dimensional algebra as simplices are in geometry', according to Leinster [14, p.216]. Opetopes and opetopic higher categories were introduced by Baez and Dolan in the seminal paper [1], and the theory has been developed further by Hermida-Makkai-Power [9], Leinster [14], Cheng [2], [3], [4], [5], and others. It is in a sense a theory from scratch, compared to several other theories of higher categories which build on large bodies of preexisting machinery and experience, e.g. simplicial methods. The full potential of the opetopic approach may depend on a deeper understanding of the combinatorics of opetopes.

At the conference on *n-categories: Foundations and applications* at the IMA in Minneapolis, June 2004, much time was dedicated to opetopes, but it became clear that a concise and direct definition of opetopes was lacking, and that there was no practical way to represent higher-dimensional opetopes on the blackboard. In fact, there did not seem to exist a general method to represent concrete opetopes in any way, algebraic,

graphical, or by machine.¹ The best definitions are very abstract and not very hands-on: e.g. Leinster’s definition in terms of iterated free cartesian monads [14], or the Hermida-Makkai-Power [9] definition of opetopic sets (there called multitopic sets), followed by a theorem that this category is a presheaf category, hence characterising a category of opetopes (there called multitopes).

As to graphical representations of opetopes in low dimensions, the current method is based on a polytope interpretation of opetopes (which is at the origin of the terminology: the word ‘opetope’ comes from ‘operation’ and ‘polytope’). Leinster [14, §7.4] has constructed a geometric realisation functor which provides support for this interpretation, although the polytopes in general cannot be piece-wise linear objects in Euclidean space. Moreover, geometrical objects in dimension higher than 3 are inherently difficult to represent graphically, and currently one resorts to Lego-like drawings in which the individual faces of the polytopes are drawn separately, with small arrows as a recipe to indicate how they are supposed to fit together.

The goal of this paper is to come closer to the combinatorics. Our initial idea was to represent an opetope as a tree with some circles, which we now call *constellations*. This works in dimension 4 (cf. 1.11 below), but it does not seem to be sufficient to capture the possible opetopes in dimension 5 and higher. Pursuing the idea, what we eventually found was a representation in terms of a sequence of trees with circles, and in fact it is basically the notion of metatree originally proposed by Baez and Dolan. That notion was never really developed, though: in the original paper [1] the claim that metatrees could express opetopes was not really substantiated, and in the subsequent literature there seems to be no mention of the metatree notion. The presence of circles makes a conceptual difference, and it also reveals a certain shortcoming in the original notion of metatree, related to units (cf. 1.21).

We hasten to point out that our notion of opetope coincides with the notion due to Leinster [14] (cf. the explicit comparison culminating in Theorem 3.16), not with the original Baez-Dolan definition: we work consistently with non-planar trees, which means our opetopes are ‘un-ordered’ like abstract geometric objects, whereas the original Baez-Dolan opetopes come equipped with an ordering of their faces. In our version, the planar aspect is only a particular feature of low dimensional opetopes.

While our opetopes agree with Leinster’s, the description we provide is completely elementary and does not even make reference to category theory. We think that our

¹ In fact a method *does* exist for algebraic/mechanical representation: Hermida-Makkai-Power [9, final section] explain how any opetope (there called multitope) in arbitrary dimension can be serialised into a string of hash signs and stars, with two sorts of brackets. We shall not go any further into that notation, but just to illustrate its flavour, here is the representation of the 3-opetope in 1.9:

$$\lrcorner\lrcorner\# \lrcorner\lrcorner\# \lrcorner\lrcorner\# \lrcorner\lrcorner\#[\star] \lrcorner\lrcorner\# \lrcorner\lrcorner\#[\star] \lrcorner\lrcorner\# \lrcorner\lrcorner\#[\star] \lrcorner\lrcorner\#[\#] \lrcorner\lrcorner\#[\#] \lrcorner\lrcorner\#[\star] \lrcorner\lrcorner\#[\star] \lrcorner\lrcorner\#[\#]$$

We refer to [9] for instructions on how to parse this.

description can serve as the famous ‘5-minute definition’ that was previously missing, and that it can provide a convenient tool for communicating opetopical ideas. We also indicate how our approach is well-suited for machine manipulation.

Opetopes were introduced to parametrise higher-dimensional substitution operations. Surprisingly, opetopes arise also in another way, namely from computads and higher-dimensional pasting theory, and we wish to mention that a very different combinatorial approach has been developed in this setting by Palm [15]. A *computad* is a strict ω -category which is dimension-wise free. This notion was devised by Street [18] as a tool for describing higher-dimensional compositions in strict n -categories. In the works of Johnson [10] and Power [16], [17], different combinatorial and topological representations of computads (called pasting schemes) were given, starting from Bénabou’s pasting diagrams for 2-categories and the dual graphical language of string diagrams. The subtleties encountered are related with the fact that the category of computads is not a presheaf category. A computad is called *many-to-one* if the codomain of every indeterminate in dimension $k + 1$ is itself an indeterminate (in dimension k). Harnik, Makkai and Zawadowski [8] established an equivalence of categories between many-to-one computads and multitopic sets. In particular, the category of many-to-one computads is a presheaf category. Palm [15] has given a purely combinatorial description of this presheaf category. He introduces a notion of *dendrotopes*, certain decorated Hasse diagrams, and shows that dendrotopic sets (their presheaves) are equivalent to many-to-one computads. Hence, by the theorems of Harnik-Makkai-Zawadowski and Hermida-Makkai-Power, dendrotopes should correspond to opetopes. However, a direct combinatorial comparison has not been given at this time.

Let us briefly outline the organisation of the exposition. In the first section we give the definition of opetopes in a direct combinatorial way, without reference to category theory. The crucial ingredient is the correspondence between *non-planar* trees and nestings of circles: an opetope is merely a sequence of such correspondences, with an initial condition. We give the definition in two steps: first the elementary ‘5-minute definition’ with examples, then we develop the involved notions of trees and constellations more formally and compare with Baez-Dolan metatrees. It is possible to jump directly from the ‘5-minute definition’ to Section 5, where the same elementary and hands-on approach is pursued to describe in detail how to compute sources and targets of opetopes, and how to compose them. However, such a reading would ignore the theoretical justification for the definitions and constructions.

In Section 2 we review some basic facts about polynomial functors, notably their graphical interpretation which is the key point to relate the formal constructions with explicit combinatorics.

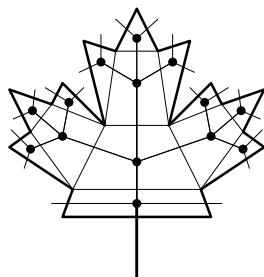
Section 3 forms the theoretical heart of this work: we give an easy account of the Baez-Dolan slice construction in the setting of polynomial monads. From the graphical

description of polynomial functors we see that the Baez-Dolan construction is about certain decorated trees. The double Baez-Dolan construction gives trees decorated with trees, subject to complicated compatibility conditions. We show that these compatibility conditions are completely encoded by drawing circles in trees. Iterating the Baez-Dolan construction involves the correspondence between trees and nestings, and it readily follows (Theorem 3.13) that the opetopes defined in Section 1 arise precisely as types for the polynomial monads produced by iterating the Baez-Dolan construction, starting from the trivial monad. We compare the polynomial Baez-Dolan construction with Leinster's version of the Baez-Dolan construction, and conclude (Theorem 3.16) that our notion of opetope agrees with Leinster's [14].

In the short Section 4, we observe a suspension operation for opetopes, and define a notion of stable opetopes. The stable opetopes also form a polynomial monad, and we show this is the least fixpoint for the Baez-Dolan construction (for pointed monads).

In Section 5, we show by way of examples how the calculus of opetopes works in practice: we are concerned with computing sources and target of opetopes, and with composing them. In the Appendix we briefly describe a machine implementation of the 'calculus of opetopes' based on XML, including a mechanism for automated graphical output.

Acknowledgements. We are grateful to John Baez and Peter May for organising the very inspiring workshop in Minneapolis, and to Eugenia Cheng and Michael Makkai for patiently telling us about opetopes at that occasion. We are grateful to editors and referees for their attentiveness. We thank our respective financing institutions: the research of A. J. was supported by the NSERC; M. B. was supported by the Australian Research Council; J.-F. M. was supported by a grant from the CNR in the framework of an IMA-CNR collaboration; J. K., currently supported by grants MTM2006-11391 and MTM2007-63277 of the Spanish Ministry of Education and Science, was previously supported by a grant from the CIRGET at the UQAM and insisted on including this opetope drawing, as expression of his gratitude and admiration:

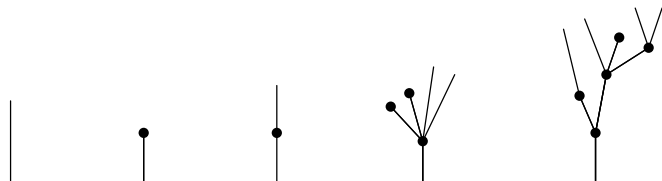


1 Opetopes

We first give the quick definition of opetope, through the notions of tree, constellation, and zoom. Afterwards we develop these notions more carefully.

The '5-minute definition' of opetope

1.1 Trees. The fundamental concept is that of a tree. Our trees are non-planar finite rooted trees with boundary: they have any number of input edges (called leaves), and have precisely one output edge (called the root edge) always drawn at the bottom. There is a partial order in which the root is the maximal element and the leaves are minimal elements. The following drawings should suffice to exemplify trees, but beware that the planar aspect inherent in a drawing should be disregarded:

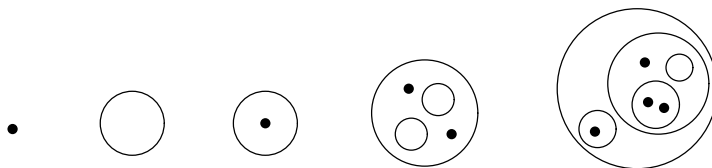


A formal definition of tree is given in 1.14. An alternative formalism is developed in [12].

1.2 Nestings. Another graphical representation of the same structure is given in terms of nested circles in the plane. We prefer to talk about nested spheres in space to avoid any idea of planarity when in a moment we combine the notion with trees. A *nesting* is a finite collection of non-intersecting spheres and dots, which either consists of a single dot (and no spheres) or has one outer sphere, containing all the other spheres and dots.

The dots of a nesting correspond to the leaves of the tree. The outer sphere corresponds to the root edge of the tree, and the special case of a nesting which consists solely of one dot corresponds to the dotless tree. The partial order is simply inclusion.

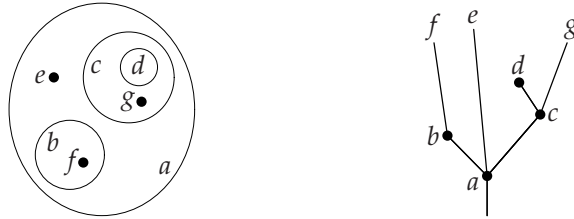
The following drawings of nestings correspond exactly to the five trees drawn above.



1.3 Correspondences. A *correspondence* between a nesting S and a tree T consists of specified bijections

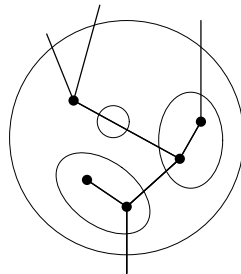
$$\begin{aligned} \text{dots}(S) &\leftrightarrow \text{leaves}(T) \\ \text{spheres}(S) &\leftrightarrow \text{dots}(T) \end{aligned}$$

respecting the partial orders. Here is a typical picture:



The bijections are indicated by the labels a, b, c, d, e, f, g .

1.4 Constellations. A *constellation* is a superposition of a tree with a nesting with common set of dots, and such that each sphere cuts a subtree. Here is an example:



More precisely, it is a configuration C of edges, dots, and spheres, such that

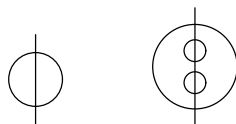
- (i) edges and dots form a tree (called the *underlying tree* of C),
- (ii) dots and spheres form a nesting (the *underlying nesting* of C),
- (iii) for each sphere, the edges and dots contained in it form a tree again.

A purely combinatorial definition of constellation is given in 1.18.

Let us briefly take a look at some degenerate examples. In a constellation without a sphere, the underlying nesting is necessarily a single dot. Hence the possibilities in this case are exhausted by the set of trees with only one dot:

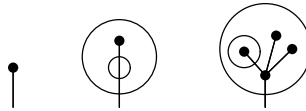


In a constellation without dots, the underlying tree must be a single edge. There must be an outer sphere, so such constellations may look like these examples:



Note that every sphere must contain a segment of a line, since there is no such thing as the empty tree.

Finally, we draw a few examples of constellations without leaves:



In 3.6 it is shown that constellations represent, in a precise sense, trees of trees, which is the reason for their importance. We want to iterate the idea of trees of trees by repeating the step of drawing spheres. To do this, we shift the nesting to a tree and iterate. In our terminology, we zoom:

1.5 Zooms. A *zoom* from constellation A to constellation B , written

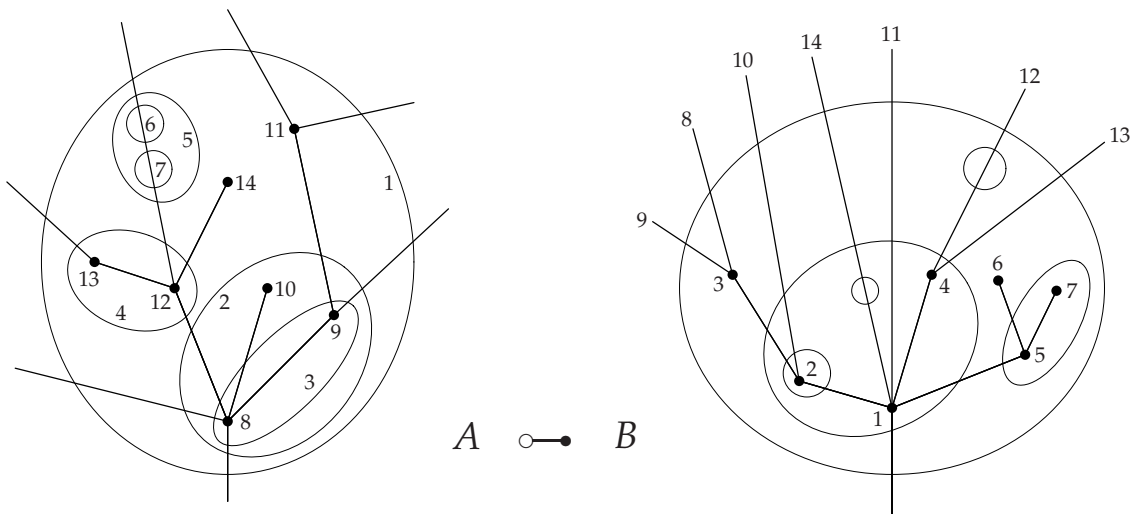
$$A \circ \bullet B,$$

is a correspondence between the underlying nesting of A and the underlying tree of B . In other words, there are specified two bijections:

$$\begin{aligned} \text{dots}(A) &\leftrightarrow \text{leaves}(B) \\ \text{spheres}(A) &\leftrightarrow \text{dots}(B) \end{aligned}$$

respecting the partial orders.

Here is an example:



The bijections are indicated with numbers.

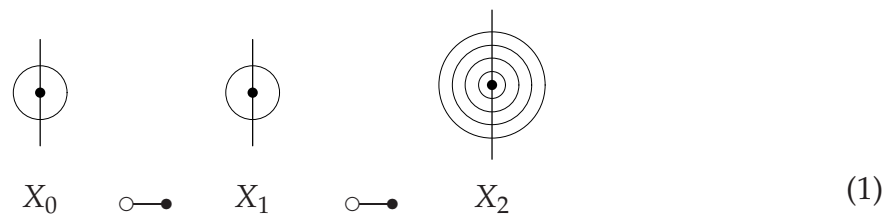
We also wish to exhibit the two most degenerate zooms:



1.6 Zoom complexes. A *zoom complex* of degree $n \geq 0$ is a sequence of zooms

$$X_0 \circ \bullet X_1 \circ \bullet X_2 \circ \bullet X_3 \circ \bullet \dots \circ \bullet X_n.$$

1.7 Opetopes. An *opetope* of dimension $n \geq 0$ is defined to be a zoom complex X of degree n starting like this:

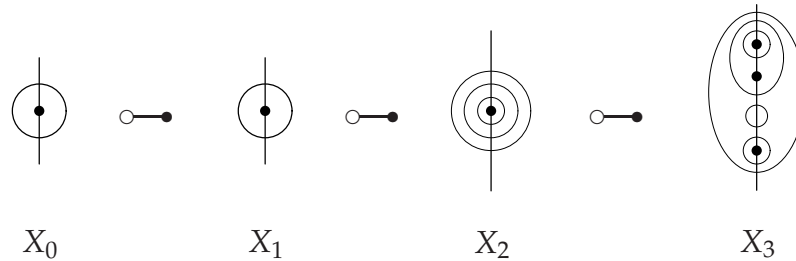


Here, X_0 and X_1 are exactly as drawn, while X_2 is described verbally as having one dot and one leaf (necessary in order to be in zoom relation with X_1), and having any finite number of linearly nested spheres. (We consider two opetopes the same if they only differ by the names of the involved elements.)

1.8 Remark. This definition of opetope should be attributed to Baez and Dolan [1] who introduced the notion of opetope in terms of a slice construction for symmetric operads (a polynomial analogue of which we shall call the *Baez-Dolan construction* (Section 3)), and offered an alternative description in terms of sequences of trees called metatrees. Definition 1.7 features important adjustments to the Baez-Dolan notion of metatree, as we shall explain in 1.21

1.9 Examples. A 0-opetope is the zoom complex \circlearrowleft (there is only one such), and a 1-opetope is the zoom complex $\circlearrowleft \circ \bullet \circlearrowleft$ (again there is only one such). The 2-opetopes are in bijection with the natural numbers, counting the linearly nested spheres in X_2 .

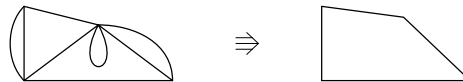
For $n \geq 3$, there are no restrictions on the constellations X_n , except to be in zoom relation with X_{n-1} . For example, if there are n spheres in X_2 , then the zoom condition forces X_3 to be a straight line with n dots on (and the bijection between spheres and dots is uniquely determined since the linear nesting of the spheres in X_2 must correspond to the linear arrangement of the dots in X_3), and any nesting can be drawn on top of that. Here is an example:



Clearly the information encoded in X_0 , X_1 and X_2 is redundant, and a 3-opetope is completely specified by a X_3 of this form: a line with dots and ‘spheres’. This is equivalent to specifying a *planar* tree. The planarity comes about because there is a line organising the dots in X_3 , which in turn is a consequence of the linear nesting of the spheres in X_2 . Here is the planar tree corresponding to the 3-opetope above:

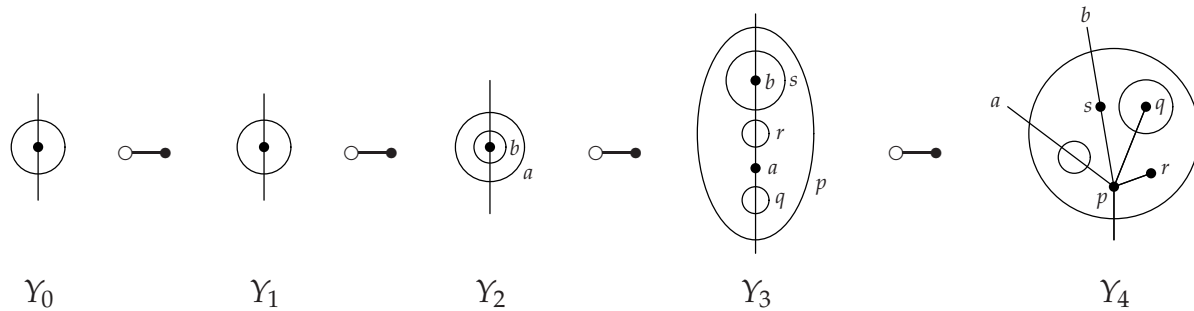


and here is how this 3-opetope would be represented in the polytope style, as in Leinster’s book [14] and in the work of Cheng:

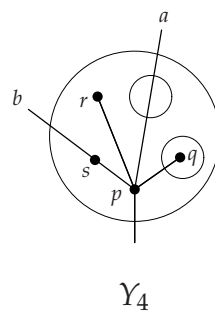


1.10 Remark. The two-step initial condition in the definition of opetope may look strange, and in any case the first two constellations are redundant in terms of information. (As we just saw, for $n \geq 3$ also X_2 is redundant, since the configuration of dots in X_3 completely determines X_2 .) The justifications for including X_0 and X_1 are first of all to cover also dimension 0 and 1 in an uniform way, and make the opetope dimension match the degree of the complex. Second, those leading \odot will play a key role in the notion of stable opetopes in 4.1. From the theoretical viewpoint, which we take up in the next section, the point is that X_0 and X_1 represent the trivial polynomial functor (the identity functor on **Set**), from which iterated application of the Baez-Dolan construction (3.1) will generate all the opetopes in higher dimension, cf. Theorem 3.13. The extra condition imposed on X_2 (the linear nesting of the spheres) is also explained by that construction. The fact that there are no extra conditions on X_n for $n \geq 3$ expresses a remarkable feature of the double Baez-Dolan construction, at the heart of this paper, namely that the double Baez-Dolan construction generates constellations, cf. Theorem 3.6.

1.11 Example. A 4-opetope is a zoom complex of degree 4 like this example:

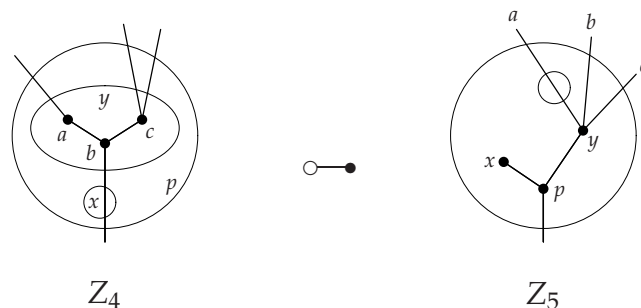


As discussed, it would be enough to indicate $Y_3 \circ \bullet Y_4$, and if we furthermore take advantage of the linear order in Y_3 and make the convention that Y_4 should be a *planar tree*, where the clockwise planar order expresses the (downwards) linear order in Y_3 , then also Y_3 is redundant, and we can represent the 4-opetope by the single constellation:



(While such economy can sometimes be practical, conceptually it is rather an obfuscation.)

1.12 Example. We finish with an example of a 5-opetope, just to point out that there is no longer any natural planar structure on the underlying trees in degree $d \geq 5$. Arguing as above, to specify a 5-opetope it is enough to specify a single zoom $Z_4 \circ \bullet Z_5$, provided we understand that the tree in Z_4 is planar (and hence allows us to reconstruct the previous constellation). Here is an example of a 5-opetope represented in this economical manner:



Formal definitions: trees and constellations

While the presented definition of opetopes is appealing in its simplicity, scrutiny of the definition raises some questions: what exactly is meant by tree? Is it a combinatorial notion? In that case, what does it mean to draw circles on a tree? And when we say ‘tree’, ‘constellation’, or ‘opetope’, do we refer to concrete specific sets with structure or do we refer to isomorphism classes of such? In this subsection we give the definitions a more formal treatment. We show in particular that the notion of constellation is purely combinatorial and does not depend on geometric realisation. Secondly, the analysis will clarify the relation to Baez-Dolan metatrees (and uncover the shortcoming with these). Thirdly, the insight provided by the formal viewpoint will be helpful for understanding the constructions in Section 3 and the calculations in Section 5.

The question of explicit-sets-with-structure versus their isomorphism classes deserves a remark before the definitions. We want to define the various notions (trees, constellations, zoom complexes, opetopes) in terms of finite sets with some structure, in order to classify as combinatorial notions. As such these objects form a proper class. On the other hand, naturally we are mostly interested in these structures up to isomorphism. Our choice will be to stick with the explicit finite-sets-with-structure as long as the objects may possess non-trivial automorphisms (which is the case for trees, constellations, and zoom complexes), but consider isomorphism classes for rigid objects like P -trees (trees decorated by a polynomial endofunctor P , as introduced in 2.8) and opetopes. Hence an opetope will be defined as a set of isomorphism classes of certain (rigid) objects (this was implicit in 1.7, and in particular there will be only a small set of them. This is in accordance with previous definitions of opetopes in the literature — in fact this issue had not previously come up since there was no combinatorial description available.

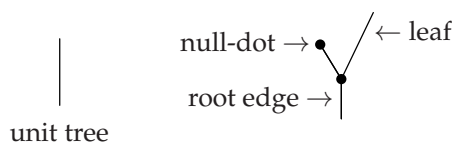
1.13 Graphs. By a *graph* we understand a pair (T_0, T_1) , where T_0 is a set, and T_1 is a set of subsets of T_0 of cardinality 2. The elements in T_0 are called *vertices*, and the elements in T_1 *edges*. An edge $\{x, y\}$ is said to be *incident* to a vertex v if $v \in \{x, y\}$. We say a vertex is of valence n if the set of incident edges is of cardinality n .

The geometric realisation of a graph is the CW-complex with a 0-cell for each vertex, and for each edge a 1-cell attached at the points corresponding to its two incident vertices.

1.14 Trees. By a *finite rooted tree with boundary* we mean a finite graph $T = (T_0, T_1)$, connected and simply connected, equipped with a pointed subset \mathbb{T} of vertices of valence 1, called the *boundary*. We will not need other kinds of trees than finite rooted trees with boundary, and we will simply call them *trees*. (An alternative tree formalism is developed in [12].)

The basepoint $t_0 \in \mathbb{T}$ is called the *output vertex*, and the remaining vertices in \mathbb{T} are

called *input vertices*. Most of the time we shall not refer to the boundary vertices at all, and graphically a boundary vertex is just represented as a loose end of the incident edge. Edges incident to input vertices are called *leaves* or *input edges* of the tree, while the unique edge incident to the output vertex is called the *root edge* or the *output edge* of the tree. The vertices in $T_0 \setminus \mathbb{T}$ are called *nodes* or *dots*; we draw them as dots. A tree may have zero dots, in which case it is just a single edge (together with two boundary vertices, which we suppress); we call such a tree a *unit tree*. Not every vertex of valence 1 needs to be a boundary vertex: those which are not are called *null-dots*.



The standard graphical representation of trees is justified by geometric realisation. Note that leaves and root are realised by half-open intervals, and we keep track of which are which by always drawing the root at the bottom.

An *isomorphism of trees* is an isomorphism of the underlying graphs preserving root and leaves. A tree can be recovered up to isomorphism by its geometric realisation. We shall frequently be interested only in the isomorphism classes. This was implicit in the '5-minute' definition.

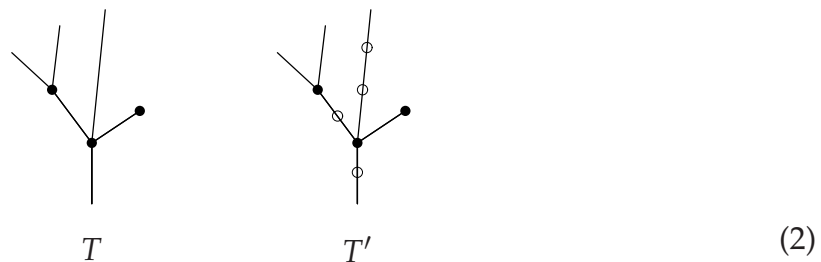
If $T = (T_0, T_1, \mathbb{T}, t_0)$ is a tree, the set T_0 has a natural poset structure $a \leq b$, in which the input vertices and null-dots are minimal elements and the output vertex is the maximal element. We say a is a *child* of b if $a \leq b$ and $\{a, b\}$ is an edge. Each dot has one output edge, and the remaining incident edges are called input edges of the dot.

1.15 Nestings and correspondences. Nestings (as in 1.2) are just another graphical representation of an abstract tree $(T_0, T_1, \mathbb{T}, t_0)$. Graphically, a *nesting* is a collection of non-intersecting spheres and dots, which either consists of a single dot (and no spheres) or has one outer sphere, containing all the other spheres and dots. We identify two nestings if there is an isotopy between them. We shall need some more terminology about nestings, expanding the dictionary between trees and nestings. A sphere that does not contain any other spheres or dots is called a *null-sphere*. These correspond exactly to the null-dots of a tree. The region bounded on the outside by a sphere S and on the inside by the dots and spheres contained in S is called a *layer*. The layers of a nesting correspond to the nodes of the tree. An inner sphere mediates between two layers just like an inner edge in a tree sits between two nodes. We will often confuse a layer with its outside bounding sphere.

1.16 Towards a combinatorial definition of constellations. In 1.4 we defined a constellation as a tree with a sphere nesting on top, more precisely as a configuration C of edges, dots, and spheres (in 3-space), such that: (i) edges and dots form a tree, (ii)

dots and spheres form a nesting, and (iii) for each sphere, the edges and dots contained in it form a tree again. This definition has a clear intuitive content, and plays an important role as convenient tool for manipulating constellations and opetopes, just like we usually manipulate trees in terms of their geometrical aspect, not in terms of abstract graphs. However, the definition depends on geometric realisation, and it is not clear at this point of our exposition that it is a rigorous notion at all. It is likely that the definition can be formalised geometrically by talking about isotopy classes of such configurations of (progressive) line segments, dots, and spheres in Euclidean space. We shall not go further into this. We wish instead to stress that the notion can be given in purely combinatorial terms. The idea is to capture the structure by specifying some bijections between the underlying tree and the tree corresponding to the nesting. For this to work it is necessary to mark the position of the null-spheres by temporarily turning them into dots. This is formalised through the notion of subdivision of trees:

1.17 Subdivision and kernels. A *linear tree* is a tree in which every dot has exactly one input edge. The unit tree is an example of a linear tree. (Up to isomorphism) there is one linear tree for each natural number. A *subdivision* of a tree T is a tree T' obtained by replacing each edge by a linear tree. We draw the new dots as white dots. Here is a picture of a tree and a subdivision:

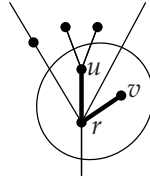


When we speak about dots of a subdivided tree we mean the union of old and new dots:

$$\text{dots}(T') = \text{blackdots}(T') + \text{whitedots}(T')$$

(note that $\text{blackdots}(T') = \text{dots}(T)$).

If T is a tree, every subset $K \subset \text{dots}(T)$ spans a full subgraph K^\dagger , where an edge of K^\dagger is an edge of T connecting two nodes of K . We call K a *kernel* if the graph K^\dagger is non-empty and connected. A kernel K spans a tree with boundary K^\dagger , whose edges are those of T adjacent to an element of K ; the dots of K^\dagger are the elements of K and the boundary vertices of K^\dagger are those vertices of K^\dagger not in K . In other words, a sphere containing exactly the dots of a kernel cuts a tree, as in condition (iii) of 1.4. In the following picture, $K = \{r, u, v\}$ is an example of a kernel, K^\dagger is indicated with fat edges, and the tree K^\dagger is what's inside the sphere:



(When we speak of kernels of a subdivided tree we refer to all dots, black and white.)

1.18 Combinatorial definition of constellation. A *constellation* $C : T \rightarrow N$ between two trees T and N is a triple $(T', \sigma_\bullet, \sigma_\circ)$, where T' is a subdivision of T , and σ_\bullet and σ_\circ are bijections

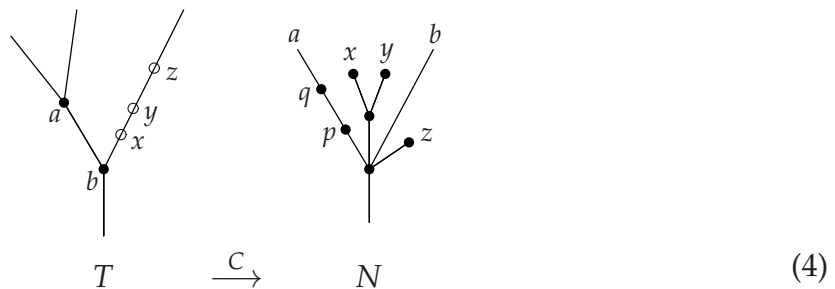
$$\begin{aligned}\sigma_\bullet &: \text{blackdots}(T') \xrightarrow{\cong} \text{leaves}(N) \\ \sigma_\circ &: \text{whitedots}(T') \xrightarrow{\cong} \text{nulldots}(N)\end{aligned}$$

such that the sum map $\sigma := \sigma_\bullet + \sigma_\circ$ satisfies the *kernel rule*:

$$\text{for each } x \in \text{dots}(N), \text{ the set } \{t \in \text{dots}(T') \mid \sigma(t) \leq x\} \text{ is a kernel in } T'. \quad (3)$$

An *isomorphism of constellations* consists of isomorphisms of the underlying (subdivided) trees compatible with the structural bijections.

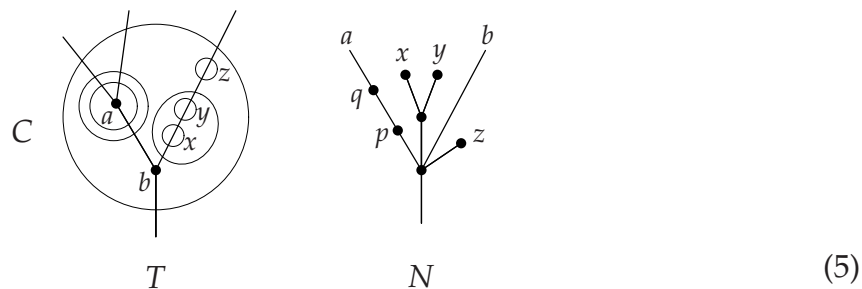
Here is a picture of a constellation in this sense:



(The white dots are not a part of T ; they represent the subdivision of T which is a part of the data constituting C .)

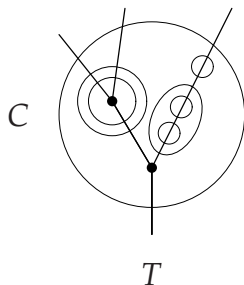
Let us compare the definition of constellation given in 1.18 with the drawings of 1.4, justifying that the latter constitute a faithful graphical representation of the former. Given a constellation according to definition 1.18, as in Figure (4), for each dot x in N that is not a null-dot, draw a sphere in T' around those dots in T' corresponding to the descendant leaves and null-dots of x in N , as in the kernel rule (3). The kernel rule tells us that this sphere cuts a tree (as in 1.4). The sphere must be drawn inside the sphere corresponding to the parent node of x (if any); this ensures that the spheres are non-intersecting and that the resulting nesting corresponds to the tree N . (Name the

spheres and white dots in T' by the corresponding dots in N .) To finish the construction, replace the white dots in T' by null-spheres.



It is now clear that the left-hand side of the picture is a constellation in the sense of 1.4. (Note that if N has no dots, then in particular it has no null-dots, so $T = T'$. Furthermore in this case N must have precisely one leaf, so T has just one dot and is therefore a constellation even without any spheres drawn.)

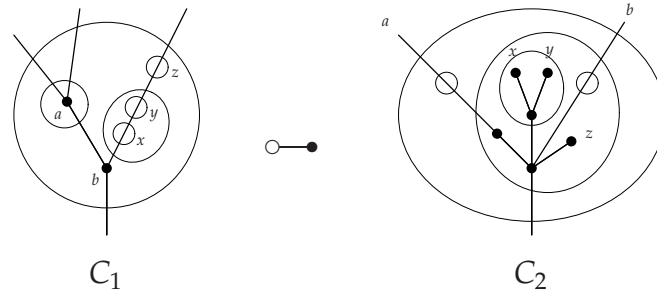
Conversely, given a constellation C in the sense of 1.4, with underlying tree T ,



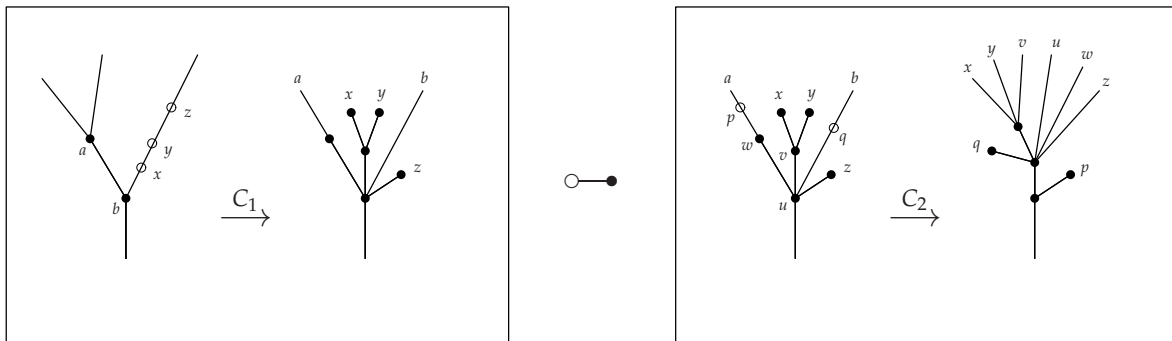
the preceding arguments can be reversed to construct a constellation according to the combinatorial definition 1.18: first draw the tree N corresponding to the underlying nesting of C (using the spheres as names for the dots in N) (this gives Figure (5)), then erase all the spheres in C except the null-spheres, and draw the null-spheres so small that they look like (white) dots — they constitute now a subdivision of T . At this point we have a constellation in the sense of 1.18: the bijections σ_\bullet and σ_\circ are already part of the correspondence between the underlying nesting of C and the tree N , and each dot $x \in \text{dots}(N)$ corresponds to a sphere in C , so the kernel rule (3) is just a reformulation of the condition that each sphere cuts a tree.

It is clear that a constellation in the sense of 1.18 can be recovered uniquely from its 1.4-interpretation.

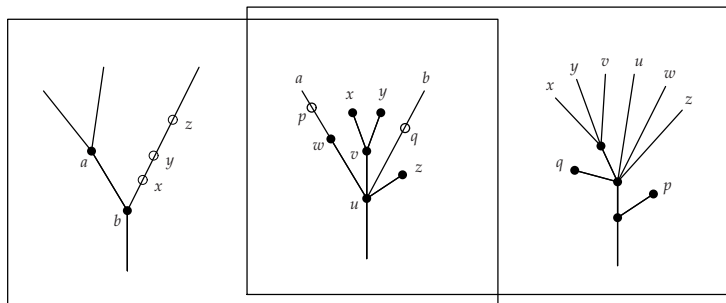
1.19 Zooms and zoom complexes, revisited. Now that the notion of constellation has been formalised, the definitions of zoom (1.5) and zoom complex (1.6) are already formal. Let us unravel these notions by plugging in the combinatorial definition of constellation (1.18). Given a zoom



the formal definition of constellation (1.18) leads to this drawing:



The defining property of zoom means the two trees in the middle coincide (modulo the subdivision, which is rather a part of the structure of C_2), so we can overlay the two constellations:



In conclusion, a zoom is a sequence of three trees connected by constellations:

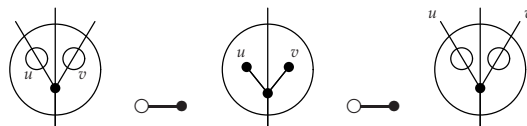
$$T_0 \xrightarrow{C_1} T_1 \xrightarrow{C_2} T_2.$$

Similarly, a zoom complex is a sequence of trees and constellations

$$T_0 \xrightarrow{C_1} T_1 \xrightarrow{C_2} T_2 \cdots T_{n-1} \xrightarrow{C_n} T_n. \tag{6}$$

An *isomorphism of zoom complexes* is a sequence of isomorphisms of constellations, compatible with the zoom bijections. In the viewpoint of (6) it is a sequence of isomorphisms of subdivided trees compatible with the structural bijections of 1.18. Note that

a zoom complex of any degree may allow non-trivial automorphisms. For example, the following zoom complex has a non-trivial involution:



1.20 Opetopes, revisited. We defined the k -opetopes to be the isomorphism classes of zoom complexes of degree k subject to an initial condition (1.7). Observe that such zoom complexes are rigid objects (i.e. have no non-trivial automorphisms). Indeed, any non-trivial automorphism of a zoom complex C induces a non-trivial automorphism already on the underlying tree of C_0 because of the structural bijections in the definition of zoom. Clearly the initial condition precludes non-trivial automorphisms in C_0 .

We saw in 1.9 that an opetope of dimension 2 can be represented by a linear tree, and an opetope of dimension 3 by a planar tree, which is the same thing as a nesting on a linear tree. In other words, an opetope of dimension 3 can be represented as a constellation $T_2 \rightarrow T_3$, where T_2 is a linear tree. In general, an opetope of dimension $n \geq 3$ can be represented by a sequence of trees and constellations

$$T_2 \xrightarrow{C_3} T_3 \xrightarrow{C_4} \dots \xrightarrow{C_n} T_n, \quad (7)$$

with T_2 a linear tree, or equivalently, as

$$C_3 \circ \bullet C_4 \circ \bullet \dots \circ \bullet C_n, \quad (8)$$

where C_3 is the constellation associated to a planar tree as in 1.9. The sequence (8) is graphically redundant compared to the sequence (7), but drawing the redundant spheres is very practical as they explicitly witness the validity of the kernel rule (3).

1.21 Relation with Baez-Dolan metatrees. The viewpoint on zoom complexes given in 1.19 provides an explicit comparison with the notion of metatree introduced by Baez and Dolan [1]. There are two important differences.

A *metatree* (cf. [1], pp. 176–177) is essentially a sequence of trees T_0, \dots, T_n not allowed to have null-dots, with specified bijections $\sigma_{\bullet, i} : \text{dots}(T_{i-1}) \xrightarrow{\sim} \text{leaves}(T_i)$ satisfying the kernel rule (3). In other words, it is the special case of a zoom complex where the trees have no null-dots, and hence there is no subdivision involved in the constellations. Null-dots represent nullary operations of the operads or polynomial monads of the Baez-Dolan construction 3.1, and nullary operations do arise. Therefore the Baez-Dolan metatrees seem to be insufficient to reflect the Baez-Dolan construction and to describe opetopes. Our zoom complexes may be what Baez and Dolan really envisaged with the notion of metatree.

The second difference is of another nature: Baez and Dolan worked with planar trees, but introduced a notion of *combed tree*, in which the leaves are allowed to cross each other in any permutation. The trees in Baez-Dolan metatrees are in fact combed. These artefacts come from working with symmetric operads. The effect on the definition of opetope is that each opetope comes equipped with an ordering of its faces. We work instead with non-planar trees and polynomial monads, and the resulting opetopes (which agree with Leinster's, cf. 3.18) are 'un-ordered' like abstract geometric objects. Planarity is revealed to be a special feature of dimension 3, cf. 1.9.

Let us remark that we think the spheres are an important conceptual device for understanding opetopes in terms of sequences of trees. Baez and Dolan stressed that a key feature of the slice construction is that operations are promoted to types, and reduction laws are promoted to operations. This two-level correspondence comes to the fore with the notion of zoom: the types are represented by the leaves, the operations are the dots, and the reduction laws are expressed by the spheres. The zoom relation shifts dots to leaves and spheres to dots.

2 Polynomial functors and polynomial monads

2.1 Polynomial functors. We recall some facts about polynomial functors. (Details for the notions needed here can be found in [7]. The manuscript [13] aims at eventually becoming a more comprehensive reference.) A diagram of sets and set maps like this

$$\begin{array}{ccc}
 & E & \xrightarrow{p} & B \\
 s \swarrow & & & & \searrow t \\
 I & & & & J
 \end{array} \tag{9}$$

gives rise to a *polynomial functor* $P : \mathbf{Set}/I \rightarrow \mathbf{Set}/J$ defined by

$$\mathbf{Set}/I \xrightarrow{s^*} \mathbf{Set}/E \xrightarrow{p_*} \mathbf{Set}/B \xrightarrow{t_!} \mathbf{Set}/J.$$

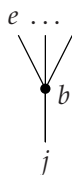
Here lowerstar and lowershriek denote, respectively, the right adjoint and the left adjoint of the pullback functor upperstar. In explicit terms, the functor is given by

$$\begin{array}{ccc}
 \mathbf{Set}/I & \longrightarrow & \mathbf{Set}/J \\
 [f : X \rightarrow I] & \longmapsto & \sum_{b \in B} \prod_{e \in E_b} X_{s(e)}
 \end{array}$$

where $E_b := p^{-1}(b)$ and $X_i := f^{-1}(i)$, and where the last set is considered to be over J via $t_!$.

We will always assume that $p : E \rightarrow B$ has finite fibres. No finiteness conditions are imposed on the individual sets I, J, E, B , nor on the fibres of s and t .

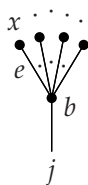
2.2 Graphical interpretation. The following graphical interpretation links polynomial functors to the tree structures of Section 1. (This interpretation is not a whim: there is a deeper relationship between polynomial functors and trees, analysed more closely in [12].) The important aspects of an element $b \in B$ are: the fibre $E_b = p^{-1}(b)$ and the element $j := t(b) \in J$. We capture these data by picturing b as a (non-planar) bouquet (also called a corolla)



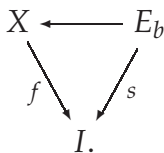
Hence each leaf is labelled by an element $e \in E_b$, and each element of E_b occurs exactly once. In virtue of the map $s : E \rightarrow I$, each leaf $e \in E_b$ acquires furthermore an implicit decoration by an element in I , namely $s(e)$.

An element in E can be pictured as a bouquet of the same type, but with one of the leaves marked (this mark chooses the element $e \in E_b$, so this description is merely an expression of the natural identification $E = \coprod_{b \in B} E_b$). Then the map $p : E \rightarrow B$ consists in forgetting this mark, and s returns the I -decoration of the marked leaf.

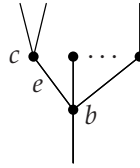
2.3 Evaluation of a polynomial functor. Evaluating the polynomial functor P on an object $f : X \rightarrow I$ has the following graphical interpretation. The elements of $P(X)$ are bouquets as above, but where each leaf is furthermore decorated by elements in X in a compatible way:



The compatibility condition for the decorations is that a leaf e may have decoration x only if $f(x) = s(e)$. The set of such X -decorated bouquets is naturally a set over J via t (return the decoration of the root edge). More formally, $P(X)$ is the set over B (and hence over J via t) whose fibre over $b \in B$ is the set of commutative triangles



2.4 Composition of polynomial functors. The composition of two polynomial functors is again polynomial; this is a consequence of distributivity and the Beck-Chevalley conditions [13]. We are mostly interested in the case $J = I$ so that we can compose P with itself. The composite polynomial functor $P \circ P$ can be described in terms of grafting of bouquets: the base set for $P \circ P$, formally described as $p_*(B \times_I E)$, is the set of bouquets of bouquets (i.e. two-level trees)



The conditions on the individual bouquets are still in force: each dot is decorated by an element in B , and for a dot with decoration b the set of incoming edges is in specified bijection with the fibre E_b . The compatibility condition for grafting is this:

Compatibility Condition: for an edge e coming out of a dot decorated c , we have

$$s(e) = t(c).$$

2.5 Morphisms. A cartesian natural transformation $u : P' \Rightarrow P$ between polynomial functors corresponds to a commutative diagram

$$\begin{array}{ccc}
 E' & \xrightarrow{p'} & B' \\
 \begin{array}{c} \swarrow s' \\ \downarrow \\ \searrow s \end{array} & \lrcorner & \begin{array}{c} \downarrow t' \\ \swarrow t \end{array} \\
 I & & J \\
 \downarrow & & \downarrow \\
 E & \xrightarrow{p} & B
 \end{array} \tag{10}$$

whose middle square is cartesian, cf. [13]. In other words, giving u amounts to giving a J -map $u : B' \rightarrow B$ together with an I -bijection $E'_{b'} \xrightarrow{\sim} E_{u(b')}$ for each $b' \in B'$.

Let $\mathbf{Poly}(I)$ denote the category whose objects are the polynomial endofunctors on \mathbf{Set}/I as in (9) and whose arrows are the cartesian natural transformations as in (10). This is a strict monoidal category under composition, and with the identity functor Id as unit object. Note that a polynomial functor always preserves cartesian squares, and (under the assumption $E \rightarrow B$ finite) sequential colimits [13].

2.6 Polynomial monads. By a *polynomial monad* we understand a polynomial endofunctor $P : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$ with monoid structure in $\mathbf{Poly}(I)$. In other words, there is specified a composition law $\mu : P \circ P \rightarrow P$ with unit $\eta : \text{Id} \rightarrow P$, satisfying the usual

associativity and unit conditions, and μ and η are cartesian natural transformations. Throughout we indicate monads by their functor part, confident that in each case it is clear what the natural-transformation part is, or explicitating it otherwise.

The composition law is described graphically as an operation of contracting two-level trees (formal compositions of bouquets) to bouquets.

We shall refer to I as the set of *types* of P , denoted $\text{typ}(P)$, and B as the set of *operations*, denoted $\text{op}(P)$. Since we have a unit, we can furthermore think of E as the set of *partial operations*, i.e. operations all of whose inputs except one are fed with a unit. The composition law can be described in terms of partial operations as a map

$$B \times_I E \rightarrow B,$$

consisting in substituting one operation into one input of another operation, provided the types match: $t(b) = s(e)$.

2.7 The free monad on a polynomial endofunctor. (See also Gambino-Hyland [6].) Given a polynomial endofunctor $P : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$, a P -set is a pair (X, a) where X is an object of \mathbf{Set}/I and $a : P(X) \rightarrow X$ is an arrow in \mathbf{Set}/I (not subject to any further conditions). A P -map from (X, a) to (Y, b) is an arrow $f : X \rightarrow Y$ giving a commutative diagram

$$\begin{array}{ccc} P(X) & \xrightarrow{P(f)} & P(Y) \\ a \downarrow & & \downarrow b \\ X & \xrightarrow{f} & Y. \end{array}$$

Let $P\text{-Set}/I$ denote the category of P -sets and P -maps. The forgetful functor $U : P\text{-Set}/I \rightarrow \mathbf{Set}/I$ has a left adjoint F , the *free P -set functor*. The monad $P^* := U \circ F : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$ is the *free monad* on P . This is a polynomial monad, and its set of operations is the set of P -trees, as we now explain.

2.8 P -trees. Let P denote a polynomial endofunctor given by $I \leftarrow E \rightarrow B \rightarrow I$. We define a P -tree to be a tree whose edges are decorated in I , whose nodes are decorated in B , and with the additional structure of a bijection for each node n (with decoration b) between the set of input edges of n and the fibre E_b , subject to the compatibility condition that such an edge $e \in E_b$ has decoration $s(e)$, and the output edge of n has decoration $t(b)$. Note that the I -decoration of the edges is completely determined by the node decoration together with the compatibility requirement, except for the case of a unit tree.

Another description is useful: a P -tree is a tree with edge set A , node set N , and node-with-marked-input-edge set N' , together with a diagram

$$\begin{array}{ccccccc}
 A & \longleftarrow & N' & \longrightarrow & N & \longrightarrow & A \\
 \alpha \downarrow & & \downarrow & \lrcorner & \downarrow \beta & & \downarrow \alpha \\
 I & \longleftarrow & E & \longrightarrow & B & \longrightarrow & I.
 \end{array}$$

Then the vertical maps α and β express the decorations, and the commutativity and the cartesian condition on the middle square express the bijections and the compatibility condition. The top row is a polynomial functor associated to a tree, and in short, a P -tree can be seen as a cartesian morphism from a tree to P in a certain category of polynomial endofunctors [12].

An *isomorphism of P -trees* is an isomorphism of trees compatible with the P -decorations. It is clear that P -trees are rigid. Denote by $\text{tr}(P)$ the set of isomorphism classes of P -trees. This is the set of formal combinations of the operations of P , i.e. obtained by freely grafting elements of B onto the leaves of elements of B , provided the decorations match (and formally adding a unit tree for each $i \in I$). The set $\text{tr}(P)$ has a natural map to I by returning the root, and it can be described as a least fixpoint for the polynomial endofunctor

$$\begin{aligned}
 \mathbf{Set}/I &\longrightarrow \mathbf{Set}/I \\
 X &\longmapsto I + P(X);
 \end{aligned}$$

as such it is given explicitly as the colimit

$$\text{tr}(P) = \bigcup_{n \in \mathbb{N}} (I + P)^n(\emptyset).$$

2.9 Explicit description of the free monad on P . A slightly more general fixpoint construction characterises the free P -set monad P^* : if A is an object of \mathbf{Set}/I , then $P^*(A)$ is a least fixpoint for the endofunctor $X \mapsto A + P(X)$. In explicit terms,

$$P^*(A) = \bigcup_{n \in \mathbb{N}} (A + P)^n(\emptyset).$$

It is the set of (isomorphism classes of) P -trees with leaves decorated in A . But this is exactly the characterisation of evaluation of a polynomial functor (2.3) with operation set $\text{tr}(P)$: let $\text{tr}'(P)$ denote the set of (isomorphism classes of) P -trees with a marked leaf, then $P^* : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$ is the polynomial functor given by

$$\begin{array}{ccc}
 & \text{tr}'(P) & \longrightarrow & \text{tr}(P) \\
 & \swarrow & & \searrow \\
 I & & & I.
 \end{array}$$

The maps are the obvious ones: return the marked leaf, forget the mark, and return the root edge, respectively. The monad structure of P^* is described explicitly in terms of grafting of trees. In a partial-composition description, the composition law is

$$\mathrm{tr}(P) \times_I \mathrm{tr}'(P) \rightarrow \mathrm{tr}(P)$$

consisting in grafting a tree onto the specified input leaf of another tree. The unit is given by $I \rightarrow \mathrm{tr}(P)$ associating to $i \in I$ the unit tree with edge decorated by i . (One can readily check that this monad is cartesian.)

3 The Baez-Dolan construction for polynomial monads

Throughout this section, we fix a polynomial monad $P : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$, represented by

$$\begin{array}{ccc} & E & \longrightarrow & B \\ & \swarrow & & \searrow \\ I & & & I \end{array}$$

We shall associate to the polynomial monad $P : \mathbf{Set}/I \rightarrow \mathbf{Set}/I$ another polynomial monad $P^+ : \mathbf{Set}/B \rightarrow \mathbf{Set}/B$. The idea of this construction is due to Baez and Dolan [1], who realised it in the settings of symmetric operads. We first give a very explicit version for polynomial monads, and show how to produce the opetopes from it by iteration, recovering the elementary definition of opetopes given in 1.7. It is the graphical interpretation of polynomial functors that allows us to extract the combinatorics. Afterwards we compare with Leinster's definition of opetopes [14, §7.1]. This is just a question of comparing our version of the Baez-Dolan construction with Leinster's; the iterative construction of opetopes is exactly the same.

Explicit construction

3.1 The Baez-Dolan construction for a polynomial monad. Starting from our polynomial monad P , we describe explicitly a new polynomial monad P^+ , the *Baez-Dolan construction* on P . The idea is to substitute into dots of trees instead of grafting at the leaves (so notice that this shift is like in a zoom relation). Specifically, define $\mathrm{tr}^\bullet(P)$ to be the set of (isomorphism classes of) P -trees with one marked dot. There is now a polynomial functor

$$\begin{array}{ccc} \mathrm{tr}^\bullet(P) & \longrightarrow & \mathrm{tr}(P) \\ \swarrow & & \searrow \\ B & & B \end{array} \quad P^+$$

where $\text{tr}^\bullet(P) \rightarrow \text{tr}(P)$ is the forgetful map, $\text{tr}^\bullet(P) \rightarrow B$ returns the bouquet around the marked dot, and $t : \text{tr}(P) \rightarrow B$ comes from the monad structure on P : it amounts to contracting all inner edges (or setting a new dot in a unit tree). Graphically:

$$\begin{array}{ccc}
 \left\{ \begin{array}{c} \text{tree with marked dot} \end{array} \right\} & \longrightarrow & \left\{ \begin{array}{c} \text{tree} \end{array} \right\} \\
 \downarrow & & \downarrow t \\
 \left\{ \begin{array}{c} \text{tree} \end{array} \right\} & P^+ & \left\{ \begin{array}{c} \text{tree} \end{array} \right\}
 \end{array} \tag{11}$$

(In this diagram as well as in the following diagrams of the same type, a symbol $\{ \Psi \}$ is meant to designate the set of *all* bouquets like this (with the appropriate decoration), but at the same time the specific figures representing each set are chosen in such a way that they match under the structure maps.) Note that since the forgetful map forgets a marked dot, the nullary operations in P^+ are precisely the unit trees $|$, one for each $i \in I$.

3.2 Monad structure on P^+ . We first compute the value of P^+ on an object $C \rightarrow B$ of \mathbf{Set}/B . Using the explicit graphical description of evaluation of a polynomial functor 2.3, we see that the result is the set of P -trees with each node decorated by an element of C , compatibly with the arity map $C \rightarrow B$ (being a P -tree means in particular that each node already has a B -decoration; these decorations must match).

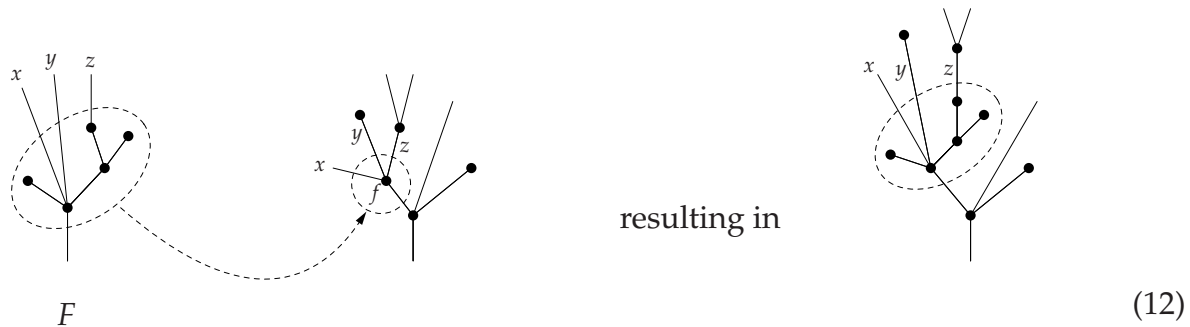
We can now compute $P^+ \circ P^+$: its set of operations is P^+ evaluated at $t : \text{tr}(P) \rightarrow B$: that's the set of (isomorphism classes of) P -trees with nodes decorated by P -trees in such a way that the total bouquet of the decorating tree matches the local bouquet of the node it decorates. Similarly, the set of 'partial operations' for $P^+ \circ P^+$ is the set of P -trees-with-a-marked-node, the marked node being decorated with a P -tree-with-a-marked-node, and the remaining nodes being decorated by P -trees.

Now the monad structure on P^+ is easy to describe: The composition law $P^+ \circ P^+ \Rightarrow P^+$ consists in substituting each P -tree into the node it decorates. The substitution can be described in terms of a partial composition law

$$\text{tr}(P) \times_B \text{tr}^\bullet(P) \rightarrow \text{tr}(P)$$

defined by substituting a P -tree into the marked dot of an element in $\text{tr}^\bullet(P)$, as indi-

cated in this figure:



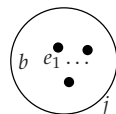
(The letters in the figure do not represent the decorations — they are rather unique labels to express the involved bijections, and to facilitate comparison with Figure (13) below.) Of course the substitution makes sense only if the decorations match. This means that $t(F)$, the ‘total bouquet’ of the tree F , is the same as the local bouquet of the node f . Formally the substitution can be described as a pushout in a category of P -trees, cf. [12].

The unit for the monad is given by the map $B \rightarrow \text{tr}(P)$ interpreting a bouquet as a tree with a single dot.

It is readily checked directly that the monad axioms hold. (Alternatively this will follow from the proof of Theorem 3.16 where P^+ is shown isomorphic to something which is a monad by construction.)

3.3 The BD construction in terms of nestings. We have described the free-monad construction and the Baez-Dolan construction in terms of trees, but of course they can equally well be described in terms of nested spheres, as we shall now explain. The interplay between these two descriptions will lead directly to opetopes as defined in Section 1. Let us stress again that trees and nestings are just different graphical expressions of the same combinatorial structure. However, some features of trees can be a little bit subtler to see in terms of nestings.

The basic operations, the elements in B , are configurations of a sphere with dots inside:

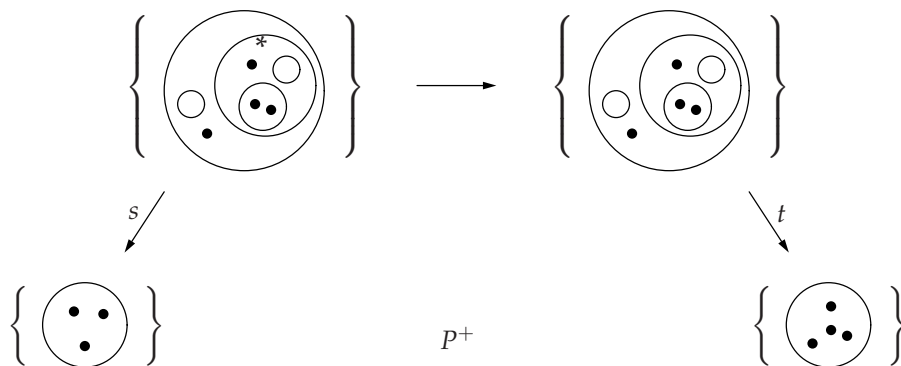


We call such a thing a *layer*. The set of dots inside the sphere is in bijection with the set E_b , and via $s : E \rightarrow I$ these dots also carry an implicit decoration by elements in I , the input types. The label j on the outside of the sphere represents $t(b)$, the output. We put the label b on the inside of the sphere it decorates, since it mediates between the

input devices (the dots) and the output device (the sphere), just as the dot of a bouquet mediates between the inputs (the leaves) and the output.

Next, $\text{tr}(P)$ is the set of (isomorphism classes of) arbitrary P -nestings, with layers decorated in B and spheres and dots decorated in I (subject to compatibility conditions), and $\text{tr}'(P)$ is the set of (isomorphism classes of) arbitrary P -nestings (compatibly decorated) with a marked dot. The substitution law for the free monad on P is now described by substituting one P -nesting into a dot of another, provided the decorations match. (This corresponds to grafting of trees.)

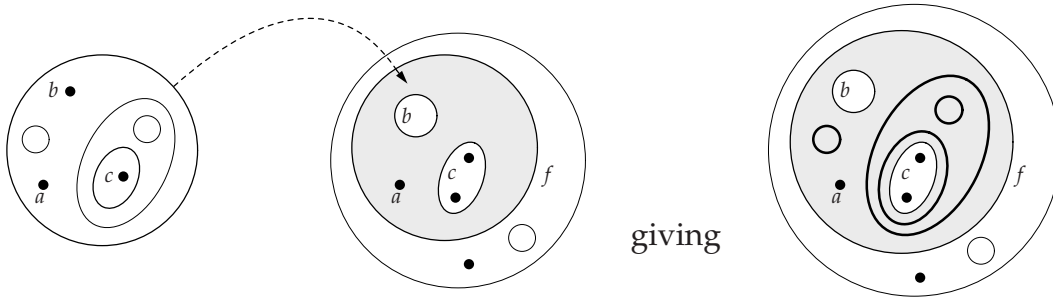
For the Baez-Dolan construction (where we now suppose P is a monad), $\text{tr}^\bullet(P)$ is the set of (isomorphism classes of) P -nestings with a marked sphere, so here is the nesting version of Figure (11):



Note that the map t consists in erasing all inner spheres, which is just the nesting equivalent of the tree operation of contracting all inner edges — this is always possible for undecorated nestings, but for this to make sense in the P -decorated case we need the monad structure on P . The map s consists in returning the *layer* determined by the marked sphere: this means the region delimited on the outside by the marked sphere itself and on the inside by its children, so the operation can also be described as taking the marked sphere and contracting each sphere inside it to a dot. (This is the nesting equivalent of the tree operation of returning the ‘local bouquet’ of a dot.)

The substitution law is perhaps less obvious in this nesting interpretation. Looking at Figure (12) we see that for trees the substitution takes place at a specified dot, and consists in replacing its ‘local bouquet’ by a more complicated tree, so the operation is about refining the tree. Correspondingly for nestings, the operation is about refining the nesting by drawing some more spheres in the specified layer. Here is the nesting

version of Figure (12):



(13)

Again, the B -decorations have not been drawn; the letters serve only to specify the bijections, and to facilitate comparison with Figure (12).

3.4 The double Baez-Dolan construction (slice-twice construction). After applying the Baez-Dolan construction once (in its tree interpretation), we have a polynomial functor $B \leftarrow \text{tr}^\bullet(P) \rightarrow \text{tr}(P) \rightarrow B$ which is a monad for the operation of substituting one tree into a dot of another tree (subject to some book-keeping). Applying the construction a second time we get

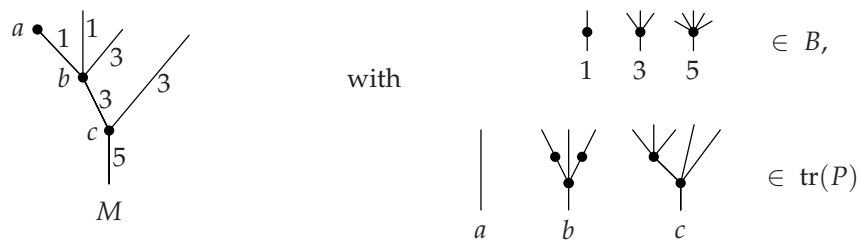
$$\begin{array}{ccc}
 \text{tr}^\bullet(P^+) & \longrightarrow & \text{tr}(P^+) \\
 \swarrow s & & \searrow t \\
 \text{tr}(P) & P^{++} & \text{tr}(P)
 \end{array}$$

Let us spell out the details. Unwinding the definitions, a P^+ -tree is a tree M whose dots are decorated by P -trees, and whose edges are decorated by elements in B , and with a specified bijection, for each node n with decorating P -tree T , between the set of input edges of n and the set of dots in T . The decoration of such an input edge must be exactly the corresponding dot in T , interpreted as an element in B , and the output edge of a dot decorated by T must be decorated by the total bouquet of T (i.e. the element of B obtained by contracting all inner edges of T using the monad structure of P). The description of the elements in $\text{tr}^\bullet(P^+)$ is similar, but with one node in M marked. The map $\text{tr}^\bullet(P^+) \rightarrow \text{tr}(P)$ returns the P -tree decorating the marked node.

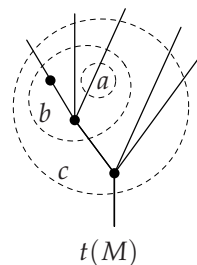
The map $\text{tr}(P^+) \rightarrow \text{tr}(P)$ involves the monad law for P^+ . Namely, we contract each inner edge of M , by composing the two P -trees decorating the adjacent dots. According to the composition law for P^+ , this means substituting the upper decorating P -tree into the designated dot of the lower decorating P -tree. (The designated dot is the one corresponding to the edge of M we are contracting, and the substitution makes sense because of the compatibility requirement of the decoration of M .) In other words, this

P -tree is obtained by successively substituting all the decorating P -trees into each other according to the recipe specified by the tree M .

Here is a drawing illustrating the notion of P^+ -tree:



And here is the result of applying t to it:



Here the dashed spheres are drawn to indicate how the original P -trees a , b , and c were substituted into each other: the inner spheres represent the ‘scars’ of the two substitutions, a into a certain node of b , and b into a certain node of c . The outer sphere represents the tree c , corresponding to the ‘root dot’ of M . Altogether we see a constellation whose underlying nesting is precisely M , and whose underlying tree is a P -tree.

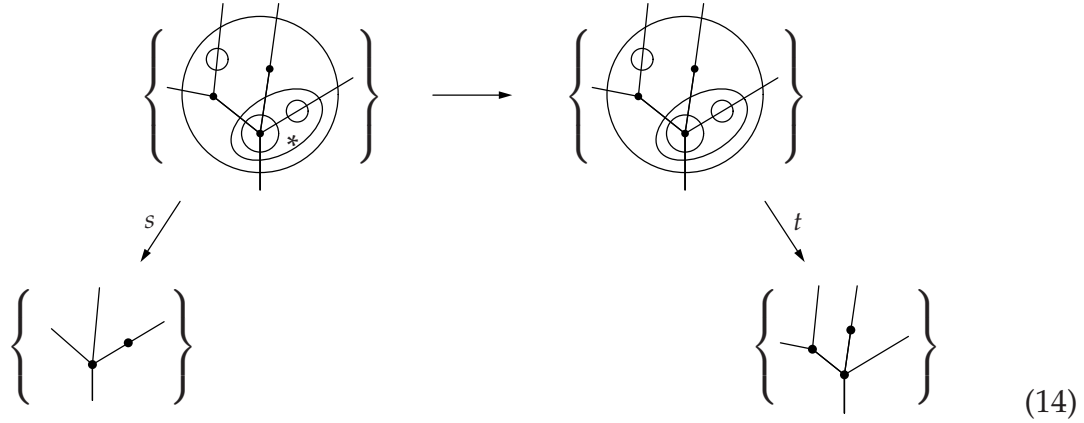
This is general: the elements in $\text{tr}(P^+)$ are obtained by successive substitutions of P -trees into nodes of a P -tree, and if for each such substitution we keep track of the surgery via the scar it left — that’s a sphere in the tree — we obtain a P -constellation. This is the content of the following theorem which also tells us that the P^+ -tree can be recovered from the P -constellation.

3.5 The P -constellation monad. By a P -constellation we mean a constellation whose underlying tree is a P -tree. Let $\text{const}(P)$ denote the set of isomorphism classes of P -constellations (note that P -constellations are rigid objects). Similarly, let $\text{const}^\circ(P)$ denote the set of isomorphism classes of P -constellations with a marked layer.

Define a polynomial endofunctor by

$$\begin{array}{ccc}
 \text{const}^\circ(P) & \longrightarrow & \text{const}(P) \\
 \swarrow & & \searrow \\
 \text{tr}(P) & & \text{tr}(P)
 \end{array}$$

Graphically,



The structure maps are: t returns the underlying tree of a constellation, and s returns the tree contained in the marked layer. The monad structure consists in substituting one constellation into the marked layer of another, provided of course their decorations match.

3.6 Theorem. *There is a natural bijection $\text{tr}(P^+) = \text{const}(P)$. This bijection is compatible with the structure maps described above, yielding an isomorphism of polynomial monads*

$$\begin{array}{ccc}
 \text{const}^\circ(P) & \xrightarrow{\quad} & \text{const}(P) \\
 \swarrow & \lrcorner & \searrow \\
 \text{tr}(P) & & \text{tr}(P) \\
 \swarrow & & \searrow \\
 \text{tr}^\bullet(P^+) & \xrightarrow{\quad} & \text{tr}(P^+)
 \end{array} \tag{15}$$

Proof. From P -constellation to P^+ -tree. Given a constellation C , we first get an abstract tree M by taking the tree corresponding to the underlying nesting of C , cf. 1.3. Let L denote the set of layers, and S the set of spheres and dots. To each layer we associate its outside sphere (the output sphere), hence a map $L \rightarrow S$. Let \bar{L} denote the set of layers with a marked child, and consider the forgetful map to L ; finally there is the obvious map $\bar{L} \rightarrow S$ returning the marked child. These maps,

$$S \leftarrow \bar{L} \rightarrow L \rightarrow S$$

is the polynomial functor associated to the tree M as in 2.8. We must now decorate this tree by P^+ , i.e., provide a diagram

$$\begin{array}{ccccccc}
 S & \xleftarrow{\quad} & \bar{L} & \xrightarrow{\quad} & L & \xrightarrow{\quad} & S \\
 \alpha \downarrow & & (3) \downarrow \gamma & \lrcorner & (2) \downarrow \beta & & (1) \downarrow \alpha \\
 B & \xleftarrow{\quad} & \text{tr}^\bullet(P) & \xrightarrow{\quad} & \text{tr}(P) & \xrightarrow{\quad} & B.
 \end{array}$$

To define α : to each dot of C we associate its local bouquet in the underlying P -tree of C . To each sphere of C , intuitively we can just look which edges come into it and which edge goes out, and this defines the local bouquet of a sphere. Note however that this description involves the monad structure of P , since in reality we are taking the P -tree T contained in the sphere and then contracting this tree to a single bouquet $t(T)$. The map β is defined similarly: to each layer, return the P -tree seen in that layer. This is the P -tree contained in the output sphere of the layer but with the subtrees in the children contracted (here again we use the monad structure of P). With α and β described this way, it is clear that square (1) commutes: both ways around the square amount to taking the bouquet around the output sphere of a given layer.

To define $\gamma : \bar{L} \rightarrow \text{tr}^\bullet(P)$, notice that the P -tree seen in a given layer has a node for each child sphere of the layer. So given a layer with a marked child, return the P -tree seen in this layer (as in the definition of β), with the node marked that corresponds to the child. Now (2) is commutative and cartesian by construction.

Finally, both ways around the square (3) amount to returning the bouquet of the marked child, which is the same as the local bouquet of the node in the tree-with-marked-node corresponding to the layer-with-marked-child.

From P^+ -tree to P -constellation. A P^+ -tree M is viewed as a recipe for how to glue small P -trees together to a big P -tree, the small P -trees being those that decorate the nodes of M . We refer to M as the *composition tree*. In the end the gluing loci will sit as spheres in the resulting big P -tree.

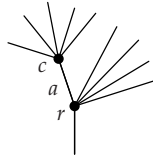
We start with the special case where the P^+ -tree M is the unit tree $|$, i.e., a single edge decorated by some bouquet $b \in B$. We need a P -constellation whose nesting corresponds to a unit tree. Hence this constellation has no spheres, and thus has just a single dot, so it amounts to giving a one-dot P -tree. Obviously we just take b itself, considered as a P -tree via the unit map for the monad.

If the composition tree M has just one dot n , this dot is decorated by a P -tree T (of a certain type). We need to provide a sphere nesting with just one sphere, and we just take T with a sphere around it.

If the composition tree M has more than one dot, then it has inner edges, and each inner edge a , say from node c down to node r represents a substitution: the P -tree T_r decorating r has a node for each input edge of r ; by the compatibility condition, the node corresponding to edge a is decorated $A = t(T_c)$, the output type of T_c . Hence it makes sense to substitute T_c into that node of T_r , cf. (12). We should perform the substitutions corresponding to all the inner edges of M . By associativity of the substitution law, we can make the substitutions edge by edge in any order.

Hence it is enough to explain what happens for a composition tree with a single

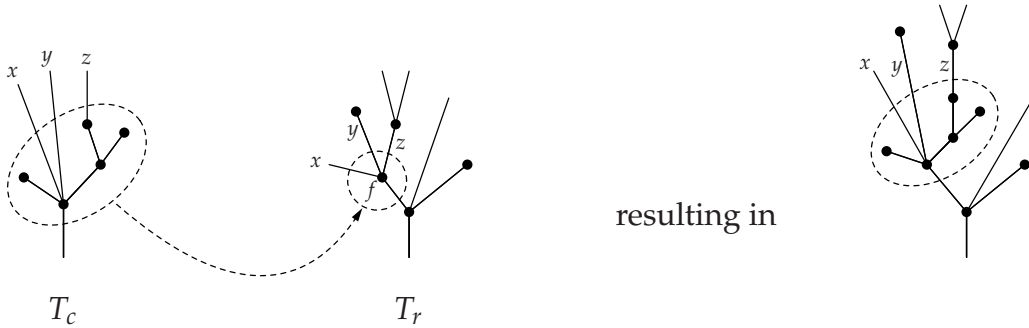
inner edge, i.e., a two-dot tree. Suppose the composition tree looks like this:



M

(16)

where node c is decorated by the P -tree T_c of output type $A \in B$, while node r is decorated by the P -tree T_r one of whose nodes f is decorated by $A \in B$. Now the substitution goes like this (cf. (12)):



T_c

T_r

resulting in

(17)

This P -tree is the underlying P -tree of the constellation we are constructing. There should be two spheres: one outer sphere (corresponding to the root edge of M) for which there is no choice, and one inner sphere corresponding to the inner edge in M . This inner sphere has to be precisely the scar of the surgery. (The remaining edges of M are leaves and correspond to dots in the constellation we are constructing.)

If the composition tree has more inner edges, each corresponding substitution will produce a sphere in the final tree, and clearly the nesting resulting from all the substitutions will correspond to the composition tree as required.

(A short remark concerning two degenerate cases: If T_c is the unit tree \mid decorated by $b \in B$, then its output type is the bouquet $b = \blacklozenge$, sitting as dot f in T_r . The effect of the substitution in this case is simply to erase the dot f , leaving a null-sphere as scar. If T_c is a one-dot tree, then we are substituting a single dot into a another dot of the same type, and the resulting tree is unchanged, but a sphere is placed around this dot, as scar of the operation. The fact that the underlying tree stays the same just says that one-dot trees are the units for the substitution law.)

It is clear from the construction that we similarly get a bijection $\text{tr}^\bullet(P^+) = \text{const}^\circ(P)$ compatible with the 'source' map and the forgetful map as in (15). Commutativity of the right-hand triangle in (15) is clear from the explicit description of the 'target' map given in 3.4. \square

To appreciate this result, note that a P^+ -tree is a complicated structure: it is a whole collection of P -trees (the decorations) satisfying a complicated set of compatibility conditions. The theorem shows that all these data can be encoded in a single P -constellation, where there are no compatibility conditions to check!

The theorem has the following interesting corollary:

3.7 Corollary. *For any polynomial monad P , any abstract tree admits a P^+ -decoration.*

In contrast, it is not true that any tree admits a decoration by a monad not of the form P^+ . For example, only linear trees can be decorated by the trivial monad.

Proof of the corollary. By the theorem, a P^+ -decoration of a tree is the same thing as a P -constellation. But every abstract nesting can appear as underlying nesting of a constellation. In fact for any P -tree, you can draw arbitrary nestings. \square

The polynomial monads of opetopes

We shall generate all the opetopes iteratively, starting from the identity monad on **Set**.

3.8 The opetope monads and the opetopes. Let P^0 denote the identity monad on **Set**,

$$\begin{array}{ccc} & 1 & \longrightarrow & 1 \\ & \swarrow & & \searrow \\ 1 & & P^0 & & 1 \end{array}$$

Let P^k denote the k th iterated Baez-Dolan construction on P^0 . By definition, the set of k -dimensional opetopes \mathbf{Z}^k is the set of types for P^k , or equivalently, for $k \geq 1$, the set of operations for P^{k-1} , or for $k \geq 2$, the set of (isomorphism classes of) P^{k-2} -trees. Finally define $\bar{\mathbf{Z}}^{k+1}$ to be the set appearing in the polynomial representation of P^k like this:

$$\begin{array}{ccc} & \bar{\mathbf{Z}}^{k+1} & \xrightarrow{p} & \mathbf{Z}^{k+1} \\ & \swarrow s & & \searrow t \\ \mathbf{Z}^k & & P^k & & \mathbf{Z}^k \end{array}$$

We define the *target* of an opetope $Z \in \mathbf{Z}^{k+1}$ to be the k -opetope $t(Z)$, and we define the *sources* of $Z \in \mathbf{Z}^{k+1}$ to be the k -opetopes $s(F)$ where F runs through the fibre $p^{-1}(Z)$.

(Sources and targets are perhaps easiest understood in terms of trees: an $(k+1)$ -opetope Z is a P^{k-1} -tree: this means its nodes are decorated by k -opetopes (the operations for P^{k-1}). These are the sources of Z . The target of Z is obtained by contracting each inner edge of the tree, correspondingly substituting the decorating k -opetopes into each other. We shall explain this in Section 5.)

Before establishing the general result reconciling this definition of opetope with the elementary combinatorial definition of 1.7, let us work out this comparison in low dimensions.

3.9 Basis for the construction. According to the definition, \mathbf{Z}^0 and \mathbf{Z}^1 are both the singleton set, in agreement with 1.7. We write $\mathbf{Z}^0 := \{ \mid \}$ and $\mathbf{Z}^1 := \{ \bullet \mid \}$, to conform with the standard graphical interpretation (cf. 2.2) of P^0 :

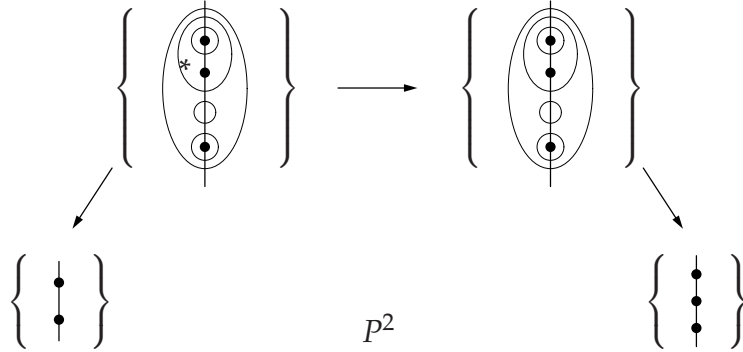
$$\begin{array}{ccc}
 \left\{ \begin{array}{c} * \\ \bullet \\ \mid \end{array} \right\} & \longrightarrow & \left\{ \begin{array}{c} \bullet \\ \mid \end{array} \right\} \\
 \swarrow & & \searrow \\
 \left\{ \begin{array}{c} \mid \\ \mid \end{array} \right\} & P^0 = \text{Id} & \left\{ \begin{array}{c} \mid \\ \mid \end{array} \right\}
 \end{array}$$

3.10 First iteration of the Baez-Dolan construction. Applying the Baez-Dolan construction to P^0 we get the polynomial monad $P^1 : \mathbf{Set} \rightarrow \mathbf{Set}$, which is nothing but the free-monoid monad $X \mapsto \sum_{n \in \mathbb{N}} X^n$. Hence $\mathbf{Z}^2 = \mathbb{N}$, in agreement with 1.7. In graphical terms, \mathbf{Z}^2 is the set of (isomorphism classes of) P^0 -trees, i.e. linear trees, and the picture is:

$$\begin{array}{ccc}
 \left\{ \begin{array}{c} \bullet \\ * \\ \bullet \\ \bullet \\ \mid \end{array} \right\} & \longrightarrow & \left\{ \begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \mid \end{array} \right\} \\
 \swarrow & & \searrow \\
 \left\{ \begin{array}{c} \bullet \\ \mid \end{array} \right\} & P^1 & \left\{ \begin{array}{c} \bullet \\ \mid \end{array} \right\}
 \end{array}$$

Note that \mathbf{Z}^2 is not yet the set of P -constellations for any P .

3.11 Second iteration of the BD construction. Performing the Baez-Dolan construction a second time defines P^2 . By Theorem 3.6, this is about setting spheres in the trees we have got, which are the linear trees. So P^2 looks like this:



So $\mathbf{Z}^3 = \text{const}(P^0)$ is the set of (isomorphism classes of) constellations whose underlying tree is linear. This is also the set of (isomorphism classes of) planar trees, in agreement with 1.7.

3.12 Third iteration of the BD construction. For the next iteration — trees of trees of trees — a new meta-device is needed, so we zoom: take the tree expression of the nesting and set spheres in it like in the previous step. More precisely, by Theorem 3.6 the set \mathbf{Z}^3 (of constellations whose underlying tree is linear) is also the set of P^1 -trees, i.e. trees with a certain compatible decoration by linear trees, and we know that to specify such a tree is just to draw the tree corresponding to the nesting, with a specified bijection: all the decorations can then be read off this bijection. Applying now the Baez-Dolan construction a third time just amounts to freely drawing spheres in these composition trees. Figure (14) serves well as illustration of P^3 , although it is not clear from the figure that the underlying tree is a P_1 -tree — but P_1 -means planar tree. In conclusion, the set of operations \mathbf{Z}^4 corresponds with the 4-opetopes defined in 1.7 and explained in 1.11.

3.13 Theorem. Let \mathbf{O}^k denote the set of k -opetopes in the sense of Definition 1.7 (isomorphism classes of degree- k zoom complexes with an initial condition). We have for $k \geq 0$ natural bijections

$$\mathbf{O}^k = \mathbf{Z}^k.$$

Proof. We already established the claim for opetopes of dimension 0, 1, 2, and 3, and proceed from here by induction. By Definition 3.8 and Theorem 3.6 we have $\mathbf{Z}^{k+3} := \text{typ}(P^{k+3}) = \text{op}(P^{k+2}) = \text{tr}(P^{k+1}) = \text{const}(P^k)$, for $k \geq 0$. So the claim is

$$\mathbf{O}^{k+3} = \text{const}(P^k) = \text{tr}(P^{k+1}),$$

and in the induction step we shall need the auxiliary statement that the spheres in the top constellation of the $(k+3)$ -opetope correspond to the spheres in the P^k -constellation (and hence to the tree in the P^{k+1} -tree).

For $k \geq 1$, suppose given a P^k -constellation. That's a P^k -tree M with some spheres — we forget the spheres for a short moment. By induction, M can be interpreted as a $(k + 2)$ -opetope W (i.e. a zoom complex of degree $k + 2$), and by the auxiliary detail, the top constellation of W has underlying nesting (composition tree) M . Now put back the spheres on M to form a zoom complex of degree $k + 3$, i.e. a $(k + 3)$ -opetope. Conversely, given a $(k + 3)$ -opetope, let M denote the underlying tree of the top constellation, and forget for a moment the spheres in M . The other constellations in the zoom complex (i.e. up to degree $k + 2$) form a $(k + 2)$ -opetope W with composition tree M . By induction, W can be interpreted as a P^k -tree, which by the auxiliary detail has underlying tree M . That is, M is a P^k -tree. Putting back the spheres on M makes it into a P^k -constellation. In both directions of the argument, it is clear that spheres correspond to spheres as required in the auxiliary detail. \square

Comparison

There exist in the literature four variations of the notion of opetope, not only in formulation but also in content: the original definition of Baez-Dolan [1], the multitopes of Hermida-Makkai-Power [9], the opetopes in terms of cartesian monads due to Leinster [14], and a modification of the Baez-Dolan notion due to Cheng [2]. The four notions have been compared by Cheng [2], [3].

We shall establish rather easily that our notion coincides with Leinster's. Our description of Leinster's sequence of cartesian monads stresses that all these monads are polynomial, and exploits the graphical calculus for polynomial functors to provide the explicit combinatorial description that was previously lacking.

3.14 The original Baez-Dolan construction. Baez and Dolan [1] described the construction first for algebras for a symmetric operad, then they applied it to symmetric operads by observing that symmetric operads are themselves algebras for some operad. This is why they had to use *symmetric* operads.

3.15 Baez-Dolan construction and definition of opetopes, according to Leinster [14, 7.1]. Let \mathcal{E} be a presheaf category, and let T be a finitary cartesian monad on \mathcal{E} . (Leinster's setup is slightly more general.) Then there is a notion of T -operad: a T -operad is a monoid in the monoidal category $\mathcal{E}/T1$ for a certain tensor product. Leinster [14, Appendix D] shows that the forgetful functor from T -operads to $\mathcal{E}/T1$ has a left adjoint, the free T -operad functor. This adjunction generates a monad which by definition is T^+ . It is clear that $\mathcal{E}/T1$ is again a presheaf category, and Leinster proves that T^+ is again a finitary cartesian monad, hence the construction can be iterated.

Leinster now defines the opetopes by starting with the identity functor T_0 on **Set** letting T_k denote the k th iterated Baez-Dolan construction, and defining the set of opetopes in dimension k to be the set of types for T_k .

Our setup is a special case of Leinster's, where \mathcal{E} is a slice of **Set**, and T is a polynomial monad. Note that polynomial functors always preserve pullbacks, and our assumption that the representing map $E \rightarrow B$ is finite amounts to T being finitary.

3.16 Theorem. *If P is a polynomial monad, the explicit polynomial Baez-Dolan construction $P \mapsto P^+$ of 3.1 coincides with Leinster's version 3.15. In particular, the opetopes defined in 1.7 and 3.8 coincide with Leinster's opetopes.*

For the proof, we first reformulate Leinster's construction and specialise it to the polynomial case.

3.17 Reformulation of Leinster's description. The reformulation removes reference to operads and the tensor product of collections. Let P be a cartesian monad on a presheaf category \mathcal{E} . Then there is a natural equivalence of categories

$$\begin{aligned} \mathbf{Cart}(\mathcal{E})/P &\simeq \mathcal{E}/P1 & (18) \\ [Q \Rightarrow P] &\mapsto [Q1 \rightarrow P1], \end{aligned}$$

where $\mathbf{Cart}(\mathcal{E})$ denotes the category of cartesian endofunctors and cartesian natural transformations. This equivalence follows readily from the fact that a cartesian natural transformation is completely determined by its value on a terminal object. The category of endofunctors over P has an obvious monoidal structure given by composition, relying on the monad structure of P : the composite of $Q \rightarrow P$ with $R \rightarrow P$ is $R \circ Q \rightarrow P \circ P \rightarrow P$ and the unit is $\text{Id} \rightarrow P$. One slick way to define the *tensor product of collections* (cf. Kelly [11]) is to transport this canonical strict monoidal structure on $\mathbf{Cart}(\mathcal{E})/P$ along the equivalence (18); operads are just monoids in the monoidal category of collections $\mathcal{E}/P1$. It follows that the free- P -operad monad on $\mathcal{E}/P1$ is equivalent to the free- P -monad monad on $\mathbf{Cart}(\mathcal{E})/P$. This monad in turn is just a matter of applying the free-monad construction on **Cart**: on an object Q this gives Q^* , and if Q is over P then Q^* is over P^* which in turn is over P in virtue of the monad structure on P . In conclusion, Leinster's Baez-Dolan construction on P consists is just the transportation along the equivalence (18) of the free-monad monad over P .

3.18 Specialisation to the polynomial case. Denote by $\mathbf{Poly}(I)$ the category whose objects are polynomial endofunctors on **Set**/ I and whose arrows are the cartesian natural transformations. Suppose P is a polynomial monad represented by

$$I \leftarrow E \rightarrow B \rightarrow I.$$

It is a basic fact [13] that any functor Q with a cartesian natural transformation to P is polynomial again, so the equivalence (18) reads

$$\begin{aligned} \mathbf{Poly}(I)/P &\simeq \mathbf{Set}/B & (19) \\ [Q \Rightarrow P] &\longmapsto [Q1 \rightarrow P1 = B]. \end{aligned}$$

The inverse equivalence takes an object $C \rightarrow B$ in \mathbf{Set}/B to the object Q in $\mathbf{Poly}(I)/P$ given by the fibre square

$$\begin{array}{ccccc}
 & E \times_B C & \longrightarrow & C & \\
 & \swarrow & & \downarrow & \searrow \\
 I & \longleftarrow & E & \longrightarrow & B \longrightarrow I.
 \end{array} \tag{20}$$

Denote by $\mathbf{PolyMon}(I)$ the category of polynomial monads on \mathbf{Set}/I , i.e. the category of monoids in $\mathbf{Poly}(I)$. The forgetful functor $\mathbf{PolyMon}(I)/P \rightarrow \mathbf{Poly}(I)/P$ has a left adjoint, the free P -monad functor, hence generating a monad $T_P : \mathbf{Poly}(I)/P \rightarrow \mathbf{Poly}(I)/P$, which we referred to above as the free- P -monad monad, and which is the BD construction on P modulo equivalence (19).

Proof of Theorem 3.16. In view of the preceding discussion, the claim of the theorem is that T_P and P^+ correspond to each other under the monoidal equivalence (19). Here P^+ denotes the explicit Baez-Dolan construction of 3.1.

We already computed the value of P^+ on an object $C \rightarrow B$ of \mathbf{Set}/B : the result is the set of P -trees with each node decorated by an element of C , compatibly with the arity map $C \rightarrow B$ (being a P -tree means in particular that each node already has a B -decoration; these decorations must match). We claim that this is the same thing as a Q -tree, where Q corresponds to $C \rightarrow B$ under equivalence (19) as in diagram (20). Indeed, since the tree is already a P -tree, we already have I -decorations on edges, as well as bijections for each node between the input edges and the fibre E_b over the decorating element $b \in B$. But if $c \in C$ decorates this same node, then the cartesian square specifies a bijection between the fibre over c and the fibre E_b and hence also with the set of input edges. So in conclusion, P^+ sends C to the set of Q -trees.

On the other hand, T_P sends the corresponding polynomial functor Q to the free monad on Q , with structure map to P given by the monad structure on P . Specifically, T_P produces from Q the polynomial monad given by

$$\begin{array}{ccc}
 \mathrm{tr}'(Q) & \longrightarrow & \mathrm{tr}(Q) \\
 \downarrow & \lrcorner & \downarrow \\
 \mathrm{tr}'(P) & \longrightarrow & \mathrm{tr}(P) \\
 \downarrow & \lrcorner & \downarrow \\
 E & \longrightarrow & B
 \end{array}$$

so the two endofunctors agree on objects. The same argument works for arrows, so the two endofunctors agree.

To see that the monad structures agree, note that the set of operations for $P^+ \circ P^+$ is the set of P -trees with nodes decorated by P -trees in such a way that the total bouquet of the decorating tree matches the local bouquet of the node it decorates. The composition law $P^+ \circ P^+ \Rightarrow P^+$ consists in substituting each tree into the node it decorates. On the other hand, to describe the monad T_P it is enough to look at the base sets, since each top set is determined as fibre product with E over B . In this optic, T_P sends B to $\text{tr}(P)$, and $T_P \circ T_P$ sends B to $\text{tr}(P^*)$, whose elements are (isomorphism classes of) P -trees with nodes decorated by P -trees, and edges decorated in I , subject to the usual compatibility conditions. Clearly the composition law $T_P \circ T_P \Rightarrow T_P$ corresponds precisely to the one we described for P^+ . For both monads, the unit is described as associating to a bouquet the corresponding one-dot tree.

In conclusion, the two constructions agree. \square

4 Suspension and stable opetopes

We introduce the notion of suspension of opetopes, define stable opetopes, and show that the accompanying monad is the least fixpoint for the Baez-Dolan construction (for pointed monads).

4.1 Suspension. The *suspension* $S(X)$ of an n -opetope X is the $(n + 1)$ -opetope defined by setting

$$\begin{aligned} S(X)_0 &:= \bigoplus \\ S(X)_{k+1} &:= X_k \quad \text{for } 0 \leq k \leq n. \end{aligned}$$

In other words, just prepend a new \bigoplus to the zoom complex, raising the indices.

The operations ‘source’, ‘target’, and ‘composition of opetopes’ all commute with suspension. Indeed, these operations are defined on the top constellations, and the repercussions down through the zoom complex can never reach the degree-1 term in the complex.

4.2 Stable opetopes. The suspension defines a map $S : \mathbf{Z}^n \rightarrow \mathbf{Z}^{n+1}$ for each $n \geq 0$. Let \mathbf{Z}^∞ denote the colimit of this sequence of maps,

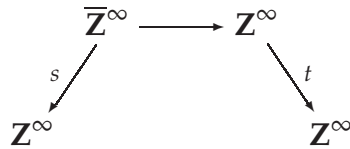
$$\mathbf{Z}^\infty = \bigcup_{n \geq 0} \mathbf{Z}^n.$$

This is the set of all opetopes in all dimensions, where we identify two opetopes if one is the suspension of the other. The elements in \mathbf{Z}^∞ are called *stable opetopes*. Note that a stable opetope has a well-defined top constellation, and that therefore the notions of source, target, and composition make sense for stable opetopes.

Define $\bar{\mathbf{Z}}^\infty := \bigcup_{n \geq 0} \bar{\mathbf{Z}}^n$, the set of stable opetopes with a marked input facet. Now consider the *polynomial monad of stable opetopes*

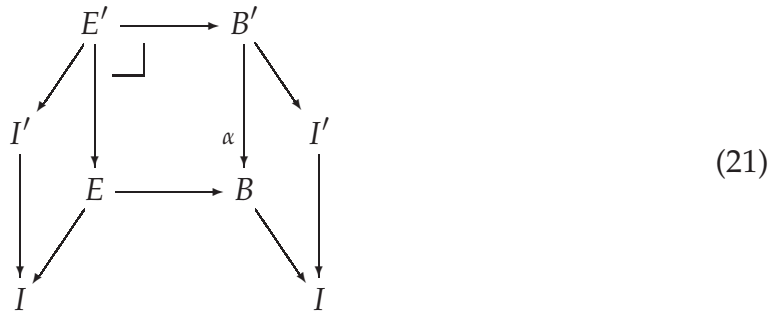
$$P^\infty : \mathbf{Set}/\mathbf{Z}^\infty \rightarrow \mathbf{Set}/\mathbf{Z}^\infty$$

defined by the diagram

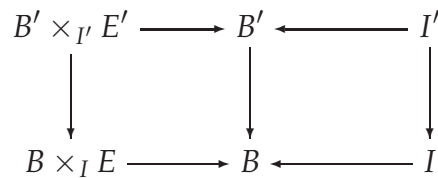


As usual, t returns the target, s returns the source, and $\bar{\mathbf{Z}}^\infty \rightarrow \mathbf{Z}^\infty$ is the forgetful map. This polynomial functor is a least fixpoint for the pointed Baez-Dolan construction, as we shall now explain.

4.3 The category of polynomial monads. Let \mathbf{PM} denote the category of all polynomial monads [7]. The arrows in this category are diagrams



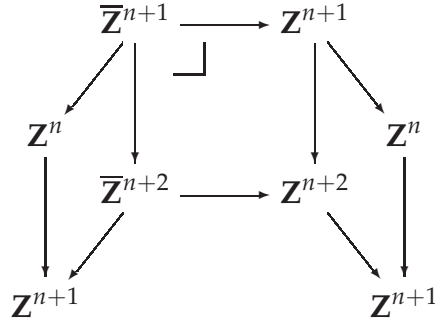
which respect the monad structure. This is most easily expressed in the partial-composition viewpoint where it amounts to requiring that these two squares commute:



The suspension map $S : \mathbf{Z}^n \rightarrow \mathbf{Z}^{n+1}$ induces an arrow in \mathbf{PM} :

$$S : P^n \rightarrow P^{n+1}$$

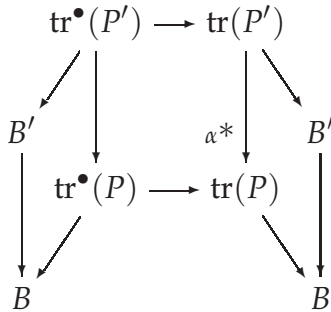
In other words, there is a natural diagram



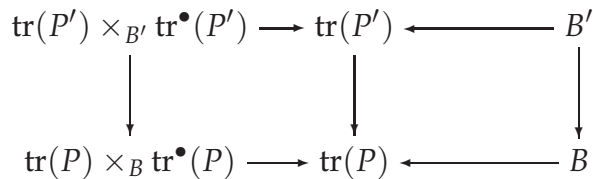
The middle square is cartesian because marking a sphere in the top constellation is independent of suspension. It is a monad map since suspension commutes with partial composition.

4.4 Proposition. *The Baez-Dolan construction is functorial: it defines a functor $BD : \mathbf{PM} \rightarrow \mathbf{PM}$.*

Proof. We have to explain what BD does on arrows (and then it will be clear that composition of arrows and identity arrows are respected). The Baez-Dolan construction on α given in (21) is:



Here $\alpha^* : \text{tr}(P') \rightarrow \text{tr}(P)$ is defined already on the level of the free-monad construction. The right-hand square commutes because α is a monad morphism. The rest is pure combinatorics, about setting marks in trees. Since α^* is defined ‘node-wise’, there is also an evident map $\text{tr}^\bullet(P') \rightarrow \text{tr}^\bullet(P)$ which makes the two other squares commute, and for which the middle square is cartesian. Finally one can check that α^* is a monad morphism:



Again this is a purely combinatorial matter: the horizontal maps are defined in terms of substituting trees into nodes of trees. Since the two rows are just two instances of this, but with different decorations, the diagram commutes. \square

4.5 Pointed polynomial monads. The Baez-Dolan functor has a rather boring least fixpoint: it is simply the initial polynomial monad $\emptyset \leftarrow \emptyset \rightarrow \emptyset \rightarrow \emptyset$. We are more interested in the notion of pointed polynomial monads and the pointed analogue of the Baez-Dolan functor.

By a *pointed polynomial monad* we understand a polynomial monad equipped with a monad map from the trivial monad

$$\begin{array}{ccc} & 1 & \longrightarrow & 1 & \\ & \swarrow & & \searrow & \\ 1 & & \text{Id} & & 1 \end{array}$$

A morphism of pointed polynomial monads is one that respects the map from Id . This defines a category \mathbf{PM}_* . If $i : \text{Id} \rightarrow M$ is a pointed polynomial monad, then $BD(M)$ is naturally pointed again, so the Baez-Dolan construction defines also a functor $\mathbf{PM}_* \rightarrow \mathbf{PM}_*$. To see this, note that by functoriality we get a map $BD(\text{Id}) \xrightarrow{BD(i)} BD(M)$. On the other hand we have $\text{Id} = P^0$, the polynomial monad of 0-opetopes, and $BD(\text{Id}) = P^1$, and the suspension map provides $\text{Id} \rightarrow BD(\text{Id})$. (Note that $P^1 : \mathbf{Set} \rightarrow \mathbf{Set}$ is the free-monoid monad.)

Now it follows readily from the standard Lambek iteration argument that

4.6 Proposition. *The polynomial monad P^∞ of stable opetopes is a least fixpoint for the Baez-Dolan construction $BD : \mathbf{PM}_* \rightarrow \mathbf{PM}_*$.*

Indeed, P^∞ can be characterised as the colimit of

$$\text{Id} \longrightarrow BD(\text{Id}) \longrightarrow BD^2(\text{Id}) \longrightarrow \dots$$

5 Calculus of opetopes — example computations

In this section we make explicit how to manipulate opetopes represented as zoom complexes. In particular we are concerned with calculating sources and target of opetopes and the operation of gluing opetopes together. A reader who has skipped Sections 2 and 3 can take the following descriptions as definitions.

In this section, by *root dot* we mean the dot adjacent to the root edge (if there are any dots).

Faces

We follow the polytope-inspired terminology for opetopes, and call their input and output devices *facets* (i.e. codimension-1 faces):

5.1 Target. The *target facet* of an n -opetope X is the $(n - 1)$ -opetope obtained by omitting the top constellation X_n and the last zoom in the zoom complex. The target is also called the *output facet*.

5.2 Sources. Let X be an n -opetope. For each sphere s in X_n , there is a *source facet* (or *input facet*), which is an $(n - 1)$ -opetope. You can think of it as the part of the zoom complex you can see by looking only through the layer determined by s , i.e., the region in X_n delimited on the outside by s itself and from the inside by the children of s .

So there are three steps in the computation of the source facet corresponding to s :

- (i) up in X_n , consider only the layer determined by s . In other words, restrict to the sphere s and contract all spheres contained in s ;
- (ii) perform certain corresponding operations on the spheres in X_{n-1} and in all lower constellations, in order to maintain the constellations in zoom relation;
- (iii) omit X_n .

In a moment we shall describe this in detail, but first it is convenient to introduce the notions of globs and drops:

5.3 Globs. An n -opetope whose top constellation X_n has precisely one sphere is called a *glob*. In this case, there is precisely one source facet, and this facet is isomorphic to the target facet. For each $(n - 1)$ -opetope F there is a unique n -glob whose target facet is F , obtained by drawing the tree corresponding to the nesting underlying F_{n-1} , and drawing a sphere around it all. This is called the glob over F . In abstract terms, it is nothing but the unit operation of type F , cf. 3.1. Hence the globs in dimension n are in natural bijection with the $(n - 1)$ -opetopes, via the target map. The term ‘glob’ comes from the polytope-style of drawing opetopes: in dimension 2 there is only one glob, which is pictured like this:



5.4 Drops. An opetope whose top constellation X_n has no spheres is called a *drop*. So a drop has no sources. Since a constellation without spheres necessarily has a unique dot, X_{n-1} has a unique sphere. Hence the target of a drop is always a glob. In particular the set of all n -drops is in bijection with the set of all $(n - 2)$ -opetopes, via the target map applied twice. Again the terminology comes from the polytope-style drawing of opetopes, where in dimension 2 one can draw the unique drop as



Notice that also in dimension 3 there is only one drop (since there is only one 1-opetope): it is the 3-opetope whose sole facet is (22).

5.5 Sphere operations. The operations involved in computing sources can be described in terms of the following sphere operations on a constellation X_i :

- Erase a sphere which is not the outer sphere.
- Draw a new sphere around a dot or a sphere.
- Contract a sphere to a dot.
- Restrict to a sphere.

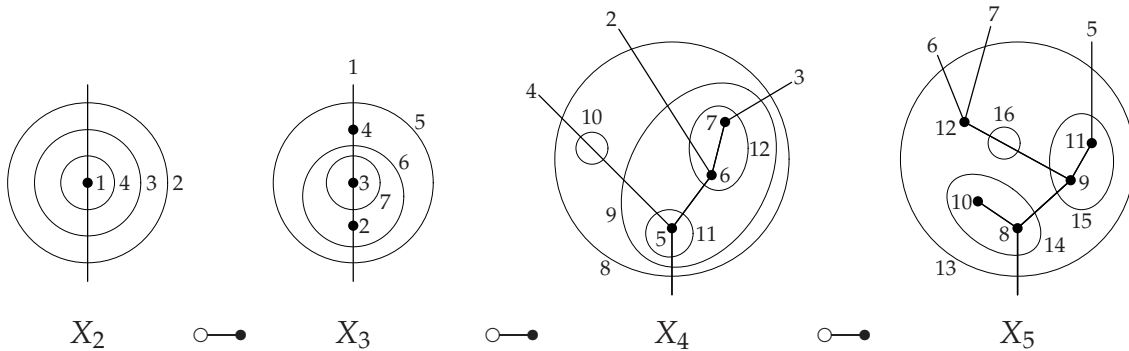
Each operation on X_i implies certain other operations on X_{i-1} , ensuring that the resulting constellations are in zoom relation, and these operations in turn imply other operations on X_{i-2} , and so on. (It is understood that the sequence of operations starts at the top constellation and propagates downwards, so we will not have to worry about consequences on X_{i+1} of an operation on X_i .)

5.6 Erasing a sphere (not the outer sphere), or drawing a new sphere around a dot or a sphere. These operations do not have any consequences in the constellation below.

5.7 Contracting a sphere to a dot. Let s be a sphere in X_i , and let T denote the tree it cuts. If there is at least one dot in T , then let r denote the root dot of T . Then we are contracting s down to r . In X_{i-1} we must erase the spheres corresponding to each non-root dot in T , and that's all. If there are no dots in T (T consists of just an edge), then we are contracting s down to a new dot which we denote s^\bullet . Since T is just a single edge, the dot s^\bullet will have a unique child c (either a dot or a leaf). In X_{i-1} we have to draw a new sphere around the sphere or dot corresponding to c .

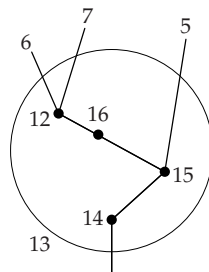
5.8 Restricting to a sphere. Let s be a sphere in X_i . Restricting to s means erasing everything outside it. The new root edge will be the root edge of the tree T cut by s , and each leaf of T will be labelled by the dot (or leaf) the edge was connecting to outside s . For each dot x that is descendant of T but not in T itself, contract the corresponding sphere x° in X_{i-1} . Finally, restrict to the sphere r° in X_{i-1} corresponding to the root dot r of T . (If T contains no dot, i.e. is just an edge, then instead of a root dot it has a unique leaf r ; in that case we are restricting to the corresponding dot r° in X_{i-1} .)

5.9 Example. We will compute the sources of the following 5-opetope:



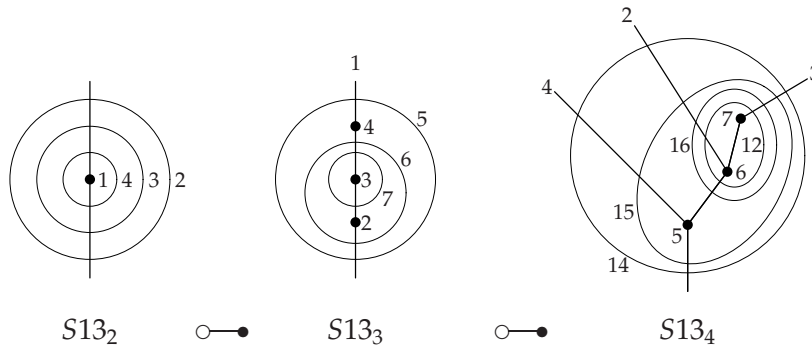
There are sources corresponding to the spheres 13, 14, 15, and 16; we will denote these source facets by S_{13} , S_{14} , S_{15} , and S_{16} .

5.10 Computation of source S_{13} . Step (i): contract 14, 15, and 16 in X_5 :

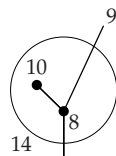


layer '13'

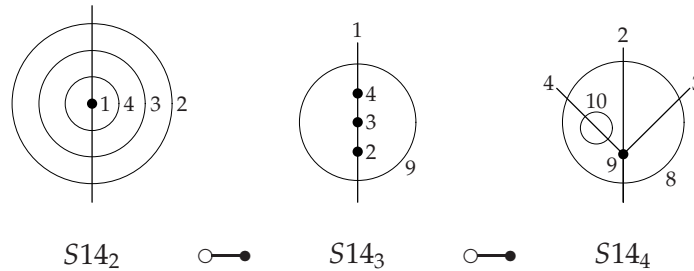
Step (ii): perform the corresponding operations in the lower constellations, according to the sphere operations rules. This means deleting spheres 10 and 11, and drawing a new sphere around sphere 12 (corresponding to the contracted 'empty' sphere 16). Finally (iii), omit the top constellation. The end result is:



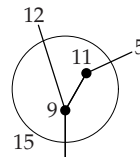
5.11 Computation of source S_{14} . Step (i): restrict to sphere 14:



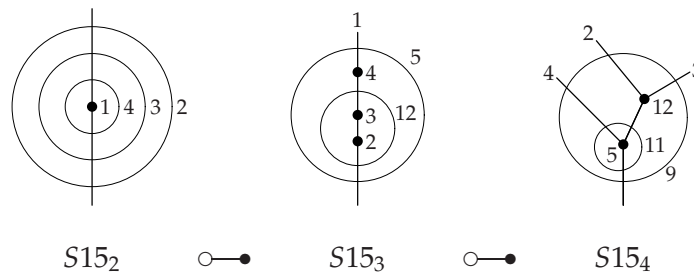
Step (ii) amounts to contracting sphere 9 in X_4 , and hence erasing sphere 6 and 7 down in X_3 . End result:



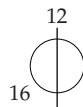
5.12 Computation of source S15. Step (i): restrict to sphere 15:



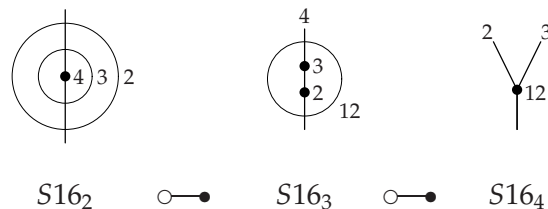
This implies (step (ii)) that in X_4 we have to restrict to sphere 9 and contract sphere 12. Down in X_3 this means erase sphere 7. End result:



5.13 Computation of source S16. Step (i): restrict to sphere 16:



Step (ii): the root of this subtree is the leaf 12, so down in X_4 we have contract sphere 12 and then restrict to the resulting dot 12. The contraction has the consequences in X_3 of erasing sphere 7 (and we rename sphere 6 to 12). Restricting to dot 12 in X_4 means restricting to sphere 12 in X_3 . Since dot 4 is a descendant which is not inside sphere 12, we have to contract sphere 4 in X_2 . End result:



Composition tree and gluing

5.14 Composition tree. The *composition tree* of an opetope is simply the tree corresponding to the nesting of the top constellation (with a specified correspondence). It concisely expresses the incidence relations among the codimension-1 faces, and how these faces are attached to each other along codimension-2 faces. We denote the composition tree of X by $\text{ct}(X)$.

In the composition tree $\text{ct}(X)$, each dot s corresponds to an input facet S (codimension-1 face). The last codimension-1 face of X , its target facet, is represented in the composition tree as the ‘total bouquet’, i.e. the bouquet obtained by contracting all inner edges (or setting a dot in the unit tree, if X is a drop).

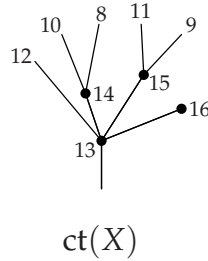
The edges in $\text{ct}(X)$ correspond to the codimension-2 faces of X : There is an incoming edge of dot s for each input facet of S , and the output edge of s represents the output facet of S . In other words, an edge linking a dot s to its parent dot p represents the codimension-2 face along which S is attached to P (the face corresponding to p): this codimension-2 face is the target facet of S and one specific source facet of P . This source is easily determined: p is a sphere in X_n and s is another sphere immediately contained in p . When computing P we contract the sphere s to a dot, hence it becomes a sphere in P_{n-1} , and so represents a source facet of P .

The leaves of $\text{ct}(X)$ correspond to the dots in the top constellation, which in turn correspond to the spheres in X_{n-1} . These are precisely the input facets of the target of X . By the preceding discussion, each of these codimension-2 faces is also the source facet of exactly one source facet of X , namely the facet S corresponding to the parent dot s of the leaf.

If there is a dot in $\text{ct}(X)$ (i.e. X is not a drop), then the root dot determines a *bottom source*, characterised also as the source facet having the same target as the target of X (corresponding to the output edge of $\text{ct}(X)$).

In summary we see that, except if X is a drop, every codimension-2 face of X occurs exactly twice as a facet of a facet. In fact, more generally, if V is a codimension- $(k+2)$ face of an opetope X , and F is a codimension- k face of X containing V , then the number of codimension- $(k+1)$ faces E such that $V \subset E \subset F$ is either 1, or 2. It is 1 if and only if F is a drop (in which case it is the drop on E (which in turn is a glob on V)).

5.15 Example (continued from 5.9). For the opetope X of the example above, the composition tree is



We see that S_{13} (corresponding to dot 13) has four input facets (corresponding to the four input edges of dot 13): the first one (leaf 12) is left vacant, its three other input facets serve as gluing locus for the output facets of S_{14} , S_{15} , and S_{16} . In turn, S_{14} and S_{15} each has two input facets (which are not in use for gluing), while S_{16} has no input facets (i.e., S_{16} is a drop). Note that the root edge represents the output facet of S_{13} .

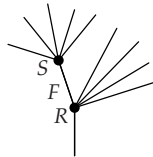
5.16 Gluing and filling. As explained in the proof of Theorem 3.6, a decorated composition tree serves as a recipe for gluing together n -dimensional opetopes S_i , producing one big n -dimensional opetope T , and finally filling the whole thing with an n -dimensional opetope X in such a way that the original opetopes S_i become the input facets of X , and T becomes the output facet.

The first part consists in producing the ‘composite’ opetope T from the S_i according to the recipe specified by the composition tree. This can be done in steps: it is enough to explain what happens when the composition tree has a single inner edge, i.e., a simple gluing. The second part (5.19) consists in constructing the filling $(n + 1)$ -opetope X .

5.17 Gluing. Given an n -opetope R with a specified source F , and another n -opetope S with target F , then their composite T is again an n -opetope, whose target is the target of R , and whose set of sources is

$$\text{sources}(S) \cup \text{sources}(R) \setminus \{F\}.$$

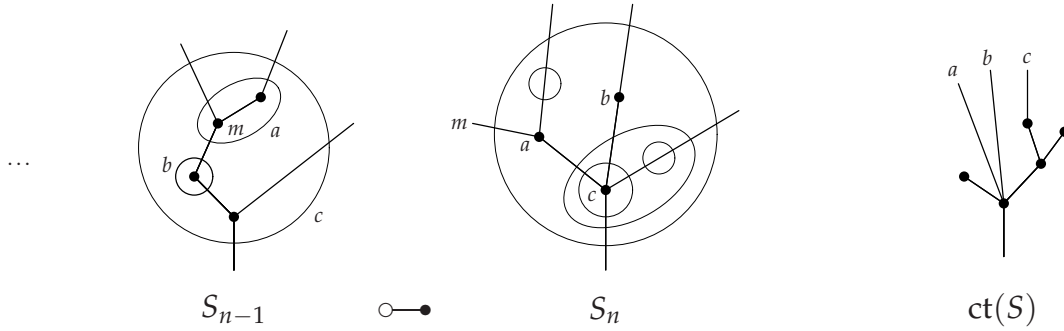
The recipe composition tree looks something like this:



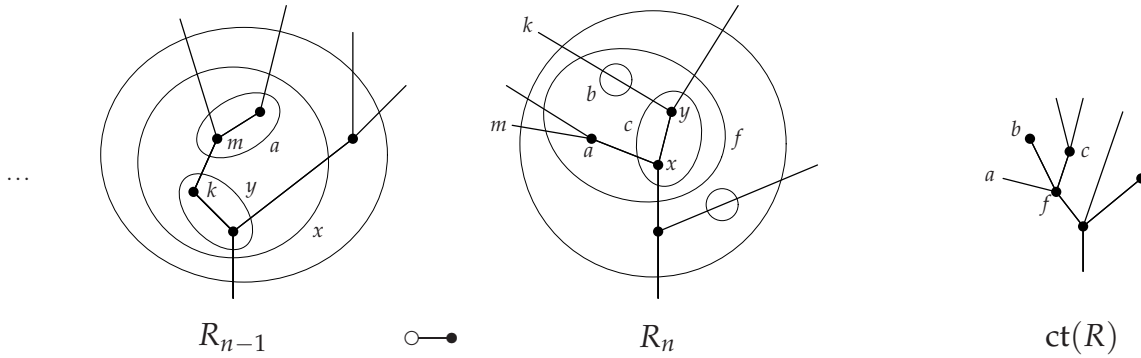
(23)

Every such situation arises as follows. Write down an arbitrary n -opetope R (but not a drop), pick one of its source facets, and write down this $(n - 1)$ -opetope F . Next we need to provide an n -opetope S having F as its target. By definition of the target map, S is obtained from F by drawing its composition tree and then drawing some arbitrary spheres in it.

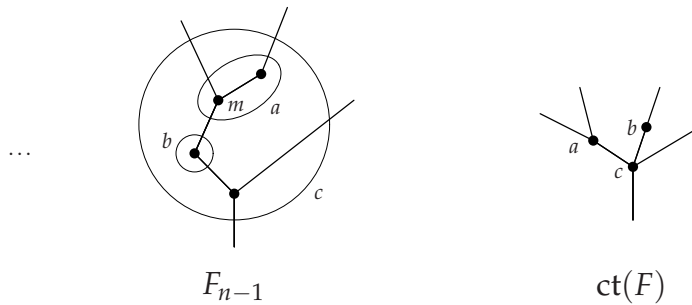
5.18 Example. Let us illustrate the situation with an example. Here is S :



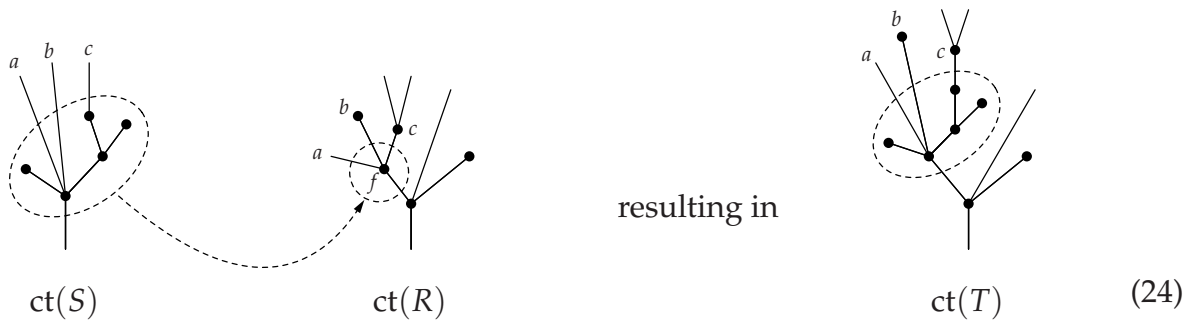
And here comes R :



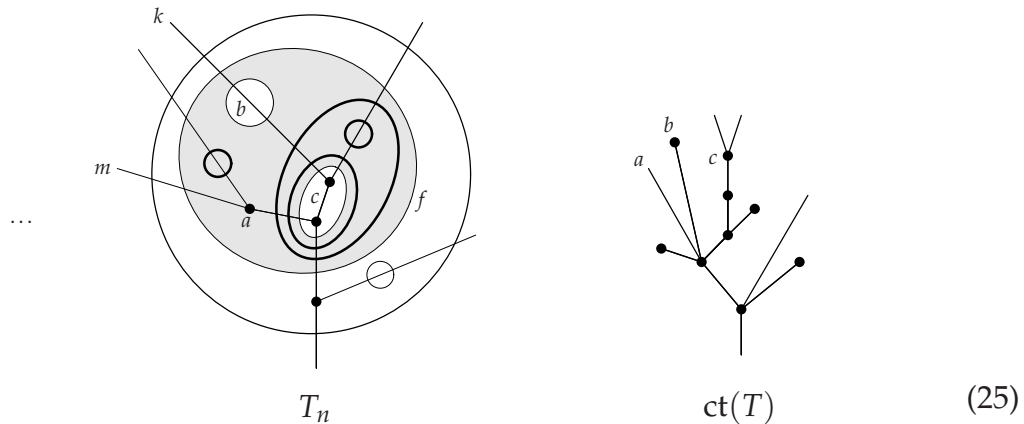
Now F is the target of S and at the same time the source of R corresponding to sphere f :



We need to construct a new n -opetope T whose target is the same as the target of R . This means that it differs from R only in the top constellation, where the configuration of spheres is different. The difference in sphere layout is expressed nicely in terms of the composition trees of S and R . The recipe prescribes that we should glue S onto the F -facet of R . In terms of the composition trees of S and R this means that we must substitute the whole tree $ct(S)$ into the node f of $ct(R)$. Since the target of S is F , this will again produce a valid decorated composition tree which will be $ct(T)$. In the current example, the situation is this:

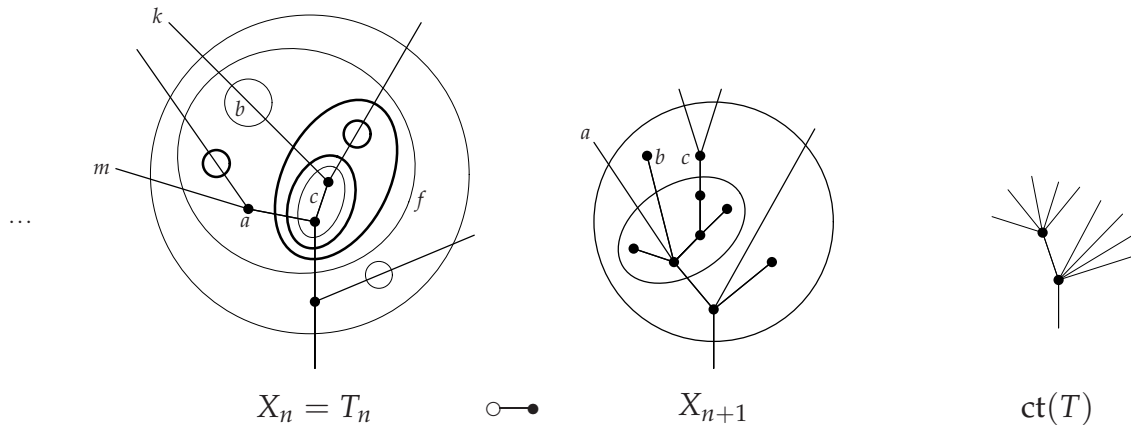


The new dots that appear in the composition tree of T specify that new spheres should be drawn in R_n in order to obtain T_n . These spheres are drawn in the layer between the sphere f and the spheres contained in f . The dot substitution performed on the composition trees is not enough information though: there is an ambiguity for the spheres corresponding to the childless dots in $ct(T)$: where should those null-spheres be drawn? But the missing bit is clearly encoded in S_n itself. In fact, substituting $ct(S)$ into the f node of $ct(R)$ is just the composition-tree expression of copying over the non-outer spheres from S_n to R_n : copy those four spheres, and paste them into the layer between the sphere f and its children. The children of f (dots and spheres immediately contained in f) are in 1–1 correspondence with the dots in S_n (since F is the target of S and the f source of R). Here is the result, with the four new spheres highlighted in fat black:



5.19 The filler. The filling $(n + 1)$ -opetope X should have T as target, so $X_k = T_k$ for $k \leq n$. The underlying tree of X_{n+1} must be the composition tree of T ; it remains to draw some spheres in this tree. These spheres are determined by the original recipe composition tree (Figure (16)): there are precisely two spheres to be drawn, corresponding to the two dots S and R in the composition tree: one sphere is the outer sphere (corresponding to the root dot R), the other sphere is the ‘scar’ of the gluing operation (corresponding to S) — this sphere was already drawn dashed in Figure (17).

So here is the final X of our running example:



It is clear from the construction that it has S and R as sources and T as target.

Appendix: Machine implementation

Our description of opetopes naturally lends itself towards machine implementation. The involved data grow only linearly with the dimension of the opetopes, and being fundamentally a tree structure, it is straightforward to encode in XML, as we shall now explain.

A.1 Trees-only representation. For the sake of machine implementation, we have adopted a variation of the trees-only representation of opetopes given in 1.20: instead of having the white dots (i.e. the null-spheres) explicitly, we let each null-dot refer to the unique child of the corresponding null-sphere in the previous constellation (be it a dot or a leaf). Now, more than one null-sphere may sit on the same edge, in which case it is not enough for the corresponding null-dots to refer to that edge. But the fact that these spheres sit on the same edge means there is induced an ordering among them, and this ordering can be expressed on the level of null-dots by letting them refer to each other in a chain, with only the last null-dot referring to something in the previous constellation (corresponding to the null-sphere farthest away from the root). This system in turn requires some careful book-keeping in connection with sphere operations, since the reference of null-dot x to a null-dot y becomes invalid if y is contracted. Keeping track of these references is not difficult, but tedious and unenlightening.

A.2 File format. XML (Extensible Mark-up Language, cf. <http://www.w3.org/XML/>) is a lot like HTML, except that you define your own tags to express a grammar. This is done in a *Document Type Definition (DTD)*. The opetope DTD looks like this:

```

<!ELEMENT opetope (constellation+)>
<!ELEMENT constellation (dot|leaf)>
<!ELEMENT dot (dot|leaf)*>
<!ELEMENT leaf EMPTY>

<!ATTLIST opetope name CDATA #REQUIRED>
<!ATTLIST constellation name CDATA #REQUIRED>
<!ATTLIST dot name CDATA #REQUIRED ref CDATA #IMPLIED>
<!ATTLIST leaf name CDATA #REQUIRED>

```

The first block declares the tags for `opetope`, `constellation`, `dot`, and `leaf`, specifying which sort of children they can have. In the second block it is specified that each tag must have a name attribute, and that the `dot` tag is also allowed an optional `ref` attribute, used only for null-dots.

Here is an XML representation of the zoom complex in Example 1.12 interpreted as a 5-opetope:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE opetope SYSTEM "opetope.dtd">
<opetope name="Z">
  <constellation name="Z4">
    <dot name="b">
      <dot name="a">
        <leaf name="1"/>
      </dot>
      <dot name="c">
        <leaf name="2"/>
        <leaf name="3"/>
      </dot>
    </dot>
  </constellation>
  <constellation name="Z5">
    <dot name="p">
      <dot name="x" ref="b"/>
      <dot name="y">
        <leaf name="a"/>
        <leaf name="b"/>
        <leaf name="c"/>
      </dot>
    </dot>
  </constellation>
  <constellation name="ct(Z)">
    <dot name="s">
      <dot name="w" ref="a"/>
      <leaf name="p"/>
      <leaf name="x"/>
      <leaf name="y"/>
    </dot>
  </constellation>
</opetope>

```

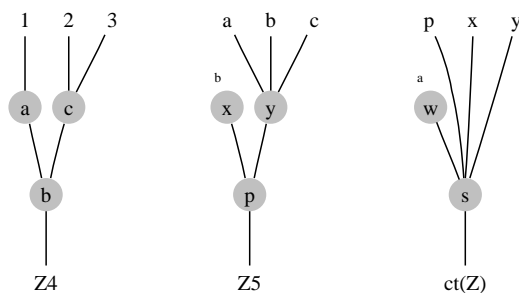
(The indentation is only for the benefit of the human reader; the XML parser ignores whitespace between the tags.) Notice how the null-dots x and w are provided with a reference to dots in the previous constellations, indicating where the corresponding spheres belong.

A.3 Scripts. The algorithms for sphere operations have been implemented in the scripting language Tcl, using the tDOM extension (cf. <http://www.tdom.org/>) for parsing and manipulating XML. There are among other things procedures for computing sources, targets, and compositions, and writing the results back to new XML files. These scripts can be run from the unix prompt, provided Tcl and the tDOM extension are available on the system. The script `computeAllFacets` takes as argument the name of an opetope XML file, and computes all its codimension-1 faces, writing the resulting opetopes to separate XML files. The script `glueOnto` takes three arguments: the bottom opetope (name of XML file), the name of the gluing locus, and the top opetope (as XML file). The result is written to a new XML file.

Precise instruction for installation and usage can be found in the readme file and manual pages accompanying the scripts. XML files for all the examples of this paper are also included, together with the XML representation of a 10-opetope with 15 input facets.

A.4 Automatic generation of graphical representation. DOT² is a language for specifying abstract graphs in terms of node-edge incidences, and generate a graphical representation of the graph, for example in PDF format. We provide a short Tcl script `opetope2pdf` which produces a dot file from an opetope XML file, and, if the dot interpreter is present on the system, also generates a pdf file. This can be helpful to get an overview of a complicated opetope and its faces, but unfortunately the output is not quite as nice as the drawings in this paper (hand-coded L^AT_EX); specifically, there is no support for drawing the spheres.

Here is what the output looks like when the script is run on the XML file listed above:



²See E. GANSNER, E. KOUTSOFIOS, and S. NORTH, *Drawing graphs with DOT*, <http://www.research.att.com/sw/tools/graphviz/dotguide.pdf>.

References

- [1] JOHN C. BAEZ and JAMES DOLAN. *Higher-dimensional algebra. III. n -categories and the algebra of opetopes*. Adv. Math. **135** (1998), 145–206. (q-alg/9702014).
- [2] EUGENIA CHENG. *Weak n -categories: opetopic and multitopic foundations*. J. Pure Appl. Algebra **186** (2004), 109–137. (math.CT/0304277).
- [3] EUGENIA CHENG. *Weak n -categories: comparing opetopic foundations*. J. Pure Appl. Algebra **186** (2004), 219–231. (math.CT/0304279).
- [4] EUGENIA CHENG. *The category of opetopes and the category of opetopic sets*. Theory Appl. Categ. **11** (2003), No. 16, 353–374 (electronic). (math.CT/0304284).
- [5] EUGENIA CHENG. *A relationship between trees and Kelly-Mac Lane graphs*. Math. Proc. Cambridge Philos. Soc. **141** (2006), 33–56. ArXiv:math/0304287.
- [6] NICOLA GAMBINO and MARTIN HYLAND. *Wellfounded trees and dependent polynomial functors*. In S. Berardi, M. Coppo, and F. Damiani, editors, *TYPES 2003*, vol. 3085 of Lecture Notes in Computer Science, pp. 210–225. Springer Verlag, Heidelberg, 2004.
- [7] NICOLA GAMBINO and JOACHIM KOCK. *Polynomial functors and polynomial monads*. Preprint, arXiv:0906.4931.
- [8] VICTOR HARNIK, MICHAEL MAKKAI, and MAREK ZAWADOWSKI. *Computads and multitopic sets*. Preprint, arXiv:0811.3215.
- [9] CLAUDIO HERMIDA, MICHAEL MAKKAI, and JOHN POWER. *On weak higher dimensional categories. I. 1–2–3*. J. Pure Appl. Algebra **154** (2000), 221–246; **157** (2001), 247–277; **166** (2002), 83–104.
- [10] MICHAEL JOHNSON. *The combinatorics of n -categorical pasting*. J. Pure Appl. Algebra **62** (1989), 211–225.
- [11] G. MAX KELLY. *On the operads of J. P. May*. Repr. Theory Appl. Categ. **13** (2005), 1–13 (electronic). (Reprint of manuscript from 1972.)
- [12] JOACHIM KOCK. *Polynomial functors and trees*. Preprint, arXiv:0807.2874.
- [13] JOACHIM KOCK. *Notes on polynomial functors*. Rough draft, 420pp. Available from <http://mat.uab.cat/~kock/cat/polynomial.html>.
- [14] TOM LEINSTER. *Higher Operads, Higher Categories*. London Math. Soc. Lecture Note Series. Cambridge University Press, Cambridge, 2004. (math.CT/0305049).

-
- [15] THORSTEN PALM. *Dendrotopic sets*. In *Galois theory, Hopf algebras, and semiabelian categories*, Fields Inst. Commun. vol. **43** (2004), 411–461. Amer. Math. Soc., Providence, RI.
- [16] A. JOHN POWER. *A 2-categorical pasting theorem*. *J. Algebra* **129** (1990), 439–445.
- [17] A. JOHN POWER. *An n -categorical pasting theorem*. In *Category theory (Como, 1990)*, vol. 1488 of *Lecture Notes in Math.*, pp. 326–358. Springer, Berlin, 1991.
- [18] ROSS STREET. *Limits indexed by category-valued 2-functors*. *J. Pure Appl. Algebra* **8** (1976), 149–181.

DEPARTAMENT DE MATEMÀTIQUES – UNIVERSITAT AUTÒNOMA DE BARCELONA – 08193
BELLATERRA (BARCELONA) – SPAIN

DÉPARTEMENT DE MATHÉMATIQUES – UNIVERSITÉ DU QUÉBEC À MONTRÉAL – CASE POSTALE
8888, SUCCURSALE CENTRE-VILLE – MONTRÉAL (QUÉBEC), H3C 3P8 – CANADA

DEPARTMENT OF MATHEMATICS, DIVISION OF ICS – MACQUARIE UNIVERSITY – NSW 2109
– AUSTRALIA

ISTITUTO PER LE APPLICAZIONI DEL CALCOLO AND INSTITUTE OF COMPLEX SYSTEMS –
NATIONAL RESEARCH COUNCIL OF ITALY – VIA DEI TAURINI 19, 00185 ROME – ITALY