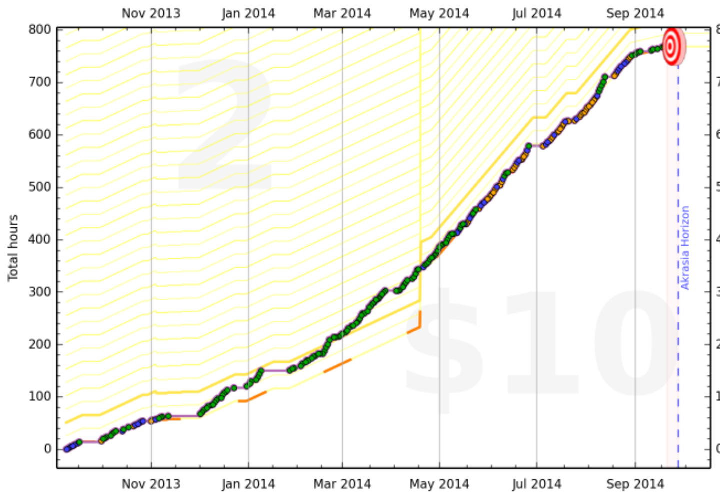
ingly small, for the simple reason that a dissertation cannot be put off until a week before it is due. $145 for the motivation to write a dissertation is quite a steal; I owe the whole Beeminder team a round of beers!



Finally, my work has been supported by the National Science Foundation under the following grants:

- NSF 1218002, CCF-SHF Small: *Beyond Algebraic Data Types: Combinatorial Species and Mathematically-Structured Programming*
- NSF 1116620, CCF-SHF Small: *Dependently-typed Haskell*
- NSF 0910500, CCF-SHF Large: *Trellys: Community-Based Design and Implementation of a Dependently Typed Programming Language*

and by the Defense Advanced Research Projects Agency under the following grant:

- DARPA Computer Science Study Panel Phase II. *Machine-Checked Metatheory for Security-Oriented Languages.*

Statement of contribution

Parts of Chapter 3, particularly the enumeration of categorical properties needed to support various species operations, were carried out in collaboration with Jacques Carette. The rest of this dissertation is my own original work.

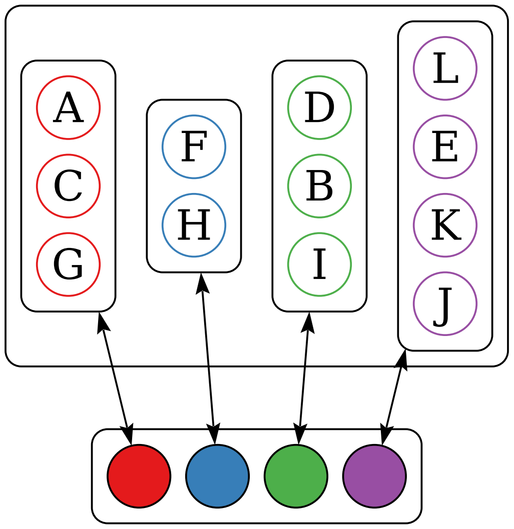


Figure 1.1: An element of $\mathbf{Set}^X$ or $\mathbf{Set}/X$

Functor categories Given two categories $\mathbb{C}$ and $\mathbb{D}$, the collection of functors from $\mathbb{C}$ to $\mathbb{D}$ forms a category (a *functor category*), with natural transformations as morphisms. This category is denoted by both of the notations $\mathbb{C} \Rightarrow \mathbb{D}$ and $\mathbb{D}^{\mathbb{C}}$, as convenient.[4] The notation $\mathbb{D}^{\mathbb{C}}$ is often helpful since intuition for exponents carries over to functor categories; for example, $\mathbb{C}^{\mathbb{D}+\mathbb{E}} \simeq \mathbb{C}^{\mathbb{D}} \times \mathbb{C}^{\mathbb{E}}$, $(\mathbb{C} \times \mathbb{D})^{\mathbb{E}} \simeq \mathbb{C}^{\mathbb{E}} \times \mathbb{D}^{\mathbb{E}}$, and so on. (In fact, this is not specific to functor categories; for example, the second isomorphism holds in any Cartesian closed category.)

Given a set X, the functor category $\mathbf{Set}^X$ (considering X as a discrete category) is equivalent to the slice category $\mathbf{Set}/X$. In particular, a functor of type $X \to \mathbf{Set}$ is an X-indexed collection of sets, whereas an object of $\mathbf{Set}/X$ is a set S with a function $f : S \to X$ labelling each element of S by some $x \in X$. Taking the preimage or *fiber* $f^{-1}(x)$ of each $x \in X$ recovers an X-indexed collection of sets; conversely, given an X-indexed collection of sets we may take their disjoint union and construct a function assigning each element of the disjoint union its corresponding element of X. Figure 1.1 illustrates the situation for $X = \{\text{red}, \text{blue}, \text{green}, \text{purple}\}$. Following the arrows from bottom to top, the diagram represents a functor $X \to \mathbf{Set}$, with each element of X mapped to a set. Following the arrows from top to bottom, the diagram represents an object in $\mathbf{Set}/X$ consisting of a set of 12 elements and an assignment of a color to each.

Equivalence of categories When are two categories "the same"? In traditional category theory, founded on set theory, there are quite a few different definitions of

[4]Traditionally the notation $[\mathbb{C}, \mathbb{D}]$ is also used, but $\mathbb{C} \Rightarrow \mathbb{D}$ is consistent with the general notation for *exponentials* explained in §1.4.2; the functor category $\mathbb{C} \Rightarrow \mathbb{D}$ is an exponential object in the Cartesian closed category of all small categories, $\mathbf{Cat}$.

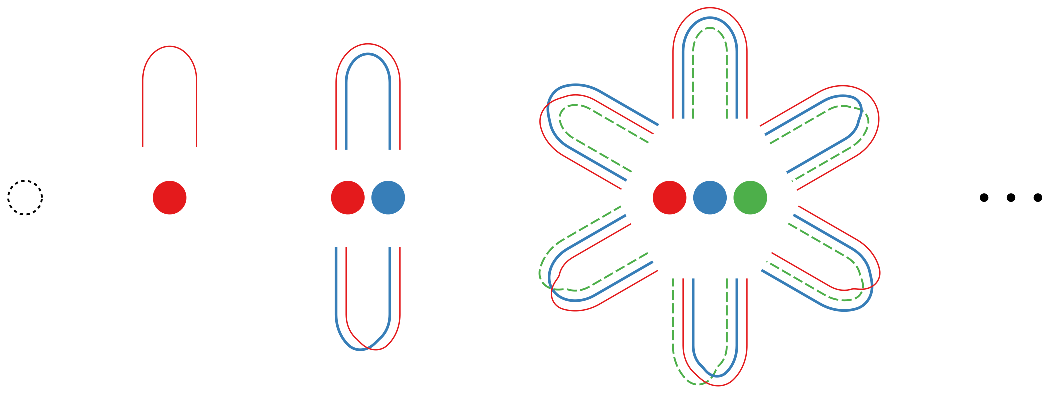


Figure 1.2: The groupoid $\mathbf{P}$

$\mathbf{B}$ does, however, have monoidal structures given by Cartesian product and disjoint union of finite sets, even though these are not a categorical product or coproduct. In particular, two bijections $\sigma_1 : S_1 \xrightarrow{\sim} T_1$ and $\sigma_2 : S_2 \xrightarrow{\sim} T_2$ naturally give rise to a bijection $(S_1 \times S_2) \xrightarrow{\sim} (T_1 \times T_2)$, which sends (s_1, s_2) to $(\sigma_1(s_1), \sigma_2(s_2))$, as well as a bijection $(S_1 \uplus S_2) \xrightarrow{\sim} (T_1 \uplus T_2)$ which sends $\mathsf{inl}\ s_1$ to $\mathsf{inl}(\sigma_1(s_1))$ and $\mathsf{inr}\ s_2$ to $\mathsf{inr}(\sigma_2(s_2))$. In fact, something more general is true:

Proposition 1.4.12. *Any monoid* $(\otimes, 1)$ *on a category* $\mathbb{C}$ *restricts to a monoid* $(\otimes^*, 1)$ *on the groupoid* $\mathbb{C}^*$*.*

Proof. There is a forgetful functor $U : \mathbb{C}^* \to \mathbb{C}$ which is the identity on both objects and morphisms. Given $X, Y \in \mathbb{C}^*$, we may define

$$X \otimes^* Y = UX \otimes UY;$$

this may be considered as an object of $\mathbb{C}^*$ since $\mathbb{C}$ and $\mathbb{C}^*$ have the same objects. Given morphisms σ and τ of $\mathbb{C}^*$, we also define

$$\sigma \otimes^* \tau = U\sigma \otimes U\tau,$$

and note that the result must be an isomorphism in $\mathbb{C}$—hence a morphism in $\mathbb{C}^*$—since all functors (here, U and $\otimes$ in particular) preserve isomorphisms.

The fact that 1 is an identity for $\otimes^*$, associativity, and the coherence conditions all follow readily from the definitions. $\boxed{\text{SDG}}$

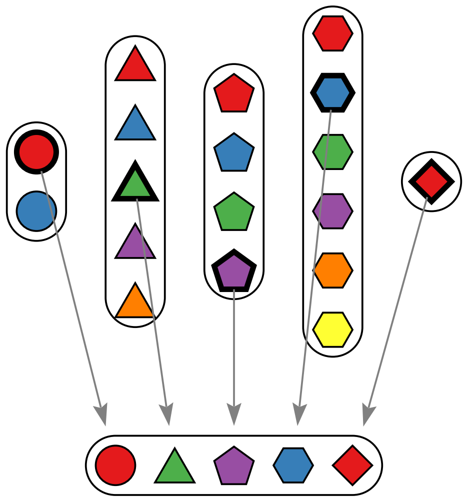


Figure 2.1: The axiom of choice

2.1 The axiom of choice (and how to avoid it)

The (in)famous *axiom of choice* (hereafter, AC) plays a central role in much of modern mathematics. In a constructive setting, however, it is problematic (§2.1.2, §2.1.3). Much effort has been expended attempting to avoid it [Makkai, 1995, 1996, 1998, Voevodsky]; in a sense, this can be seen as one of the goals of the univalent foundations program. In §2.2 and §2.4 we will see how HoTT indeed provides an excellent AC-free foundation for the mathematics we want to do. First, however, we give an introduction to AC and related issues in set theory.

2.1.1 The axiom of choice and constructive mathematics

The axiom of choice can be formulated in a number of equivalent ways. Perhaps the most well-known is

The Cartesian product of any collection of non-empty sets is non-empty. (AC)

Given a family of sets $\{X_i \mid i \in I\}$, an element of their Cartesian product is some I-indexed tuple $\{x_i \mid i \in I\}$ where $x_i \in X_i$ for each i. Such a tuple can be thought of as a function (called a *choice function*) which picks out some particular x_i from each X_i. This can be visualized (for a particularly small and decidedly finite case) as shown in Figure 2.1.

Note that AC is *independent* of the usual set theory foundations (the so-called *Zermelo-Fraenkel axioms*, or ZF), in the sense that it is consistent to add either AC or its negation to ZF. It is somewhat controversial since it has some (seemingly)

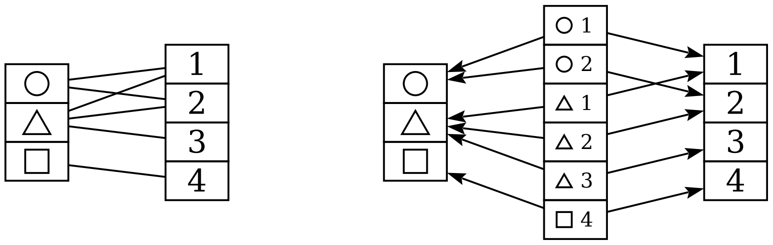


Figure 2.2: Representing a many-to-many relationship via a junction table

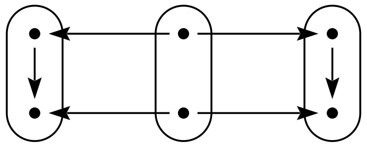
- There are two functions $\operatorname{Ob}\mathbb{C} \xleftarrow{\overleftarrow{F}} S \xrightarrow{\overrightarrow{F}} \operatorname{Ob}\mathbb{D}$ mapping specifications to objects of $\mathbb{C}$ and $\mathbb{D}$.

S, $\overleftarrow{F}$, and $\overrightarrow{F}$ together define a many-to-many relationship between objects of $\mathbb{C}$ and objects of $\mathbb{D}$. $D \in \mathbb{D}$ is called a *specified value of F at C* if there is some specification $s \in S$ such that $\overleftarrow{F}(s) = C$ and $\overrightarrow{F}(s) = D$, in which case we write $F_s(C) = D$. Moreover, D is *a value of F at C* (not necessarily a *specified* one) if there is some s for which $D \cong F_s(C)$.

The idea now is to impose additional conditions which ensure that F acts like a regular functor $\mathbb{C} \to \mathbb{D}$.

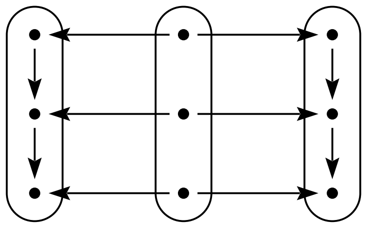
- Functors are defined on all objects; so we require each object of $\mathbb{C}$ to have at least one specification s which corresponds to it—that is, $\overleftarrow{F}$ must be surjective.

- Functors transport morphisms as well as objects. For each $s, t \in S$ (the middle of the below diagram) and each $f : \overleftarrow{F}(s) \to \overleftarrow{F}(t)$ in $\mathbb{C}$ (the left-hand side below), there must be a morphism $F_{s,t}(f) : \overrightarrow{F}(s) \to \overrightarrow{F}(t)$ in $\mathbb{D}$ (the right-hand side):



- Functors preserve identities: for each $s \in S$ we should have $F_{s,s}(id_{\overleftarrow{F}(s)}) = id_{\overrightarrow{F}(s)}$.

- Finally, functors preserve composition: for all $s, t, u \in S$ (in the middle below), $f : \overleftarrow{F}(s) \to \overleftarrow{F}(t)$, and $g : \overleftarrow{F}(t) \to \overleftarrow{F}(u)$ (the left side below), it must be the case that $F_{s,u}(f \,;\, g) = F_{s,t}(f) \,;\, F_{t,u}(g)$:



Remark. Our initial intuition was that an anafunctor should map objects of $\mathbb{C}$ to equivalence classes of objects in $\mathbb{D}$. This may not be immediately apparent from the definition, but is in fact the case. In particular, the identity morphism id_C maps to isomorphisms between specified values of C; that is, under the action of an anafunctor, an object C together with its identity morphism "blow up" into a clique. To see this, let $s, t \in S$ be two different specifications corresponding to C, that is, $\overleftarrow{F}(s) = \overleftarrow{F}(t) = C$. Then by preservation of composition and identities, we have

$$F_{s,t}(id_C) \,;\, F_{t,s}(id_C) = F_{s,s}(id_C \,;\, id_C) = F_{s,s}(id_C) = id_{\overrightarrow{F}(s)},$$

so $F_{s,t}(id_C)$ and $F_{t,s}(id_C)$ constitute an isomorphism between $F_s(C)$ and $F_t(C)$.

Remark. It is not hard to show that cliques in $\mathbb{D}$ are precisely anafunctors from $\mathbf{1}$ to $\mathbb{D}$. In fact, more is true: the class of functors $\mathbb{C} \to \operatorname{clq}\mathbb{D}$ is naturally isomorphic to the class of anafunctors $\mathbb{C} \to \mathbb{D}$ (for the proof, see Makkai [1996, pp. 31–34]).

There is an alternative, equivalent definition of anafunctors, which is somewhat less intuitive but usually more convenient to work with.

Definition 2.1.5. An anafunctor $F : \mathbb{C} \to \mathbb{D}$ is a category $\mathbb{S}$ together with a span of *functors* $\mathbb{C} \xleftarrow{\overleftarrow{F}} \mathbb{S} \xrightarrow{\overrightarrow{F}} \mathbb{D}$ where $\overleftarrow{F}$ is fully faithful and (strictly) surjective on objects.

Remark. In this definition, $\overleftarrow{F}$ must be *strictly* (as opposed to *essentially*) surjective on objects, that is, for every $C \in \mathbb{C}$ there is some $S \in \mathbb{S}$ such that $\overleftarrow{F}(S) = C$, rather than only requiring $\overleftarrow{F}(S) \cong C$. Given this strict surjectivity on objects, it is equivalent to require $\overleftarrow{F}$ to be full, as in the definition above, or to be (strictly) surjective on the class of all morphisms.

We are punning on notation a bit here: in the original definition of anafunctor, S is a set and $\overleftarrow{F}$ and $\overrightarrow{F}$ are functions on objects, whereas in this more abstract definition $\mathbb{S}$ is a category and $\overleftarrow{F}$ and $\overrightarrow{F}$ are functors. Of course, the two are closely related: given a span of functors $\mathbb{C} \xleftarrow{\overleftarrow{F}} \mathbb{S} \xrightarrow{\overrightarrow{F}} \mathbb{D}$, we may simply take the objects of $\mathbb{S}$ as the class of specifications S, and the actions of the functors $\overleftarrow{F}$ and $\overrightarrow{F}$ on

2.3 Finiteness in set theory

Finally, we can assemble the foregoing material on anafunctors and category theory in HoTT into a coherent story about *finiteness*, first using set-theoretic foundations, and then using HoTT. The material in this section and the next (other than the lemmas and theorems cited from the HoTT book) is novel.

Recall that $\mathbf{B}$ denotes the groupoid of finite sets and bijections, and $\mathbf{P}$ the groupoid of natural numbers and permutations. In classical category theory, $\mathbf{P}$ is a *skeleton* of $\mathbf{B}$—roughly, we may think of it as the result of replacing each equivalence class of isomorphic objects in $\mathbf{B}$ with a single object. In this case, we identify each equivalence class of isomorphic finite sets with a natural number *size*—size being the one property of sets which is invariant under isomorphism. The relationship between $\mathbf{B}$ and $\mathbf{P}$ is central to the concept of finiteness: passing from $\mathbf{B}$ to $\mathbf{P}$ corresponds to taking the *size* of finite sets, and passing from $\mathbf{P}$ to $\mathbf{B}$ corresponds to constructing canonical finite sets of a given size. The study of $\mathbf{B}$ and $\mathbf{P}$ is also critical for the theory of species; as we will shortly see in Chapter 3, traditional species are defined as functors $\mathbf{B} \to \mathbf{Set}$.

It is a simple result in classical category theory that every category is equivalent to its skeletons. This equivalence allows one to pass freely back and forth between functors $\mathbf{B} \to \mathbf{Set}$ and functors $\mathbf{P} \to \mathbf{Set}$, and this is often implicitly exploited in the literature on species. However, we are interested in the *computational* content of this equivalence, and it is here that we run into trouble. After the foregoing discussion of cliques and anafunctors, the idea of quotienting out by equivalence classes of isomorphic objects ought to make us squeamish—and, indeed, the proof that $\mathbf{B}$ and $\mathbf{P}$ are equivalent requires AC.

In more detail, it is easy to define a functor $[-] : \mathbf{P} \to \mathbf{B}$ which sends the natural number n to the finite set $[n]$ and preserves morphisms; defining an inverse functor $\#- : \mathbf{B} \to \mathbf{P}$, however, is more problematic. We can send each set S to its size $\#S$, but we must send each bijection $S \xrightarrow{\sim} T$ to a permutation $[\#S] \xrightarrow{\sim} [\#T]$, and there is no obvious way to pick one. For example, suppose $S = \{\mathrm{cat}, \mathrm{dog}, \mathrm{moose}\}$ and $T = \{\bigcirc, \triangle, \square\}$. Given a bijection matching each animal with its favorite shape[4], it must be sent to a permutation on $\{0, 1, 2\}$—but to which permutation should it be sent? Knowing that the size of $\{\mathrm{cat}, \mathrm{dog}, \mathrm{moose}\}$ is 3 does not tell us anything about how to match up animals with $\{0, 1, 2\}$.

Abstractly, $[-] : \mathbf{P} \to \mathbf{B}$ is fully faithful and essentially surjective (every finite set is in bijection with $[n]$ for some n); this yields an equivalence of categories, and hence an inverse functor $\#- : \mathbf{B} \to \mathbf{P}$, only in the presence of AC. More concretely, we can use AC to choose an arbitrary bijection $\varphi_S : S \xrightarrow{\sim} [\#S]$ for each finite set S, somehow matching up S with the canonical set of size $\#S$. Given $\alpha : S \xrightarrow{\sim} T$ we can then construct

$$[\#S] \xrightarrow{\varphi_S^{-1}} S \xrightarrow{\alpha} T \xrightarrow{\varphi_T} [\#T] .$$

[4]The details are left as an exercise for the reader.

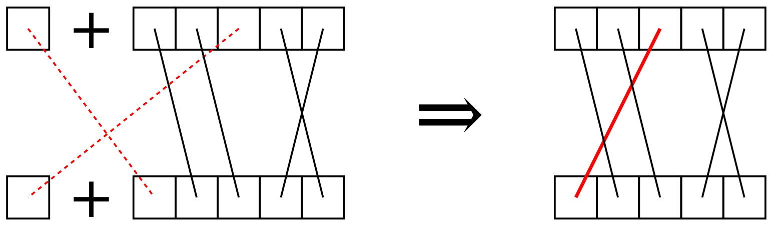


Figure 2.3: Eliminating $\top$ from both sides of an equivalence

Corollary 2.4.2. *If A and B are sets, then so is $A = B$.*

Proof. Immediate from univalence and Lemma 2.4.1. ∎

Lemma 2.4.3. *For all $n_1, n_2 : \mathbb{N}$, if $\mathsf{Fin}\ n_1 \simeq \mathsf{Fin}\ n_2$ then $n_1 = n_2$.*

Proof. The proof is by double induction on n_1 and n_2.

- If both n_1 and n_2 are zero, the result is immediate.
- The case when one is zero and the other a successor is impossible. In particular, taking the equivalence in the appropriate direction gives a function $\mathsf{Fin}\ (\mathsf{S} \dots) \to \mathsf{Fin}\ \mathsf{O}$, which can be used to produce an element of $\mathsf{Fin}\ \mathsf{O} = \bot$, from which anything follows.
- In the case when both are a successor, we have $\mathsf{Fin}\ (\mathsf{S}\ n_1') \simeq \mathsf{Fin}\ (\mathsf{S}\ n_2')$, which is equivalent to $\top + \mathsf{Fin}\ n_1' \simeq \top + \mathsf{Fin}\ n_2'$. If we can conclude that $\mathsf{Fin}\ n_1' \simeq \mathsf{Fin}\ n_2'$, the inductive hypothesis then yields $n_1' = n_2'$, from which $\mathsf{S}\ n_1' = \mathsf{S}\ n_2'$ follows immediately. The implication $(\top + \mathsf{Fin}\ n_1' \simeq \top + \mathsf{Fin}\ n_2') \to (\mathsf{Fin}\ n_1' \simeq \mathsf{Fin}\ n_2')$ is true, but not quite as straightforward to show as one might think! In particular, an equivalence $(\top + \mathsf{Fin}\ n_1' \simeq \top + \mathsf{Fin}\ n_2')$ may not match the $\top$ values with each other. As illustrated in Figure 2.3, given $e : (\top + \mathsf{Fin}\ n_1' \simeq \top + \mathsf{Fin}\ n_2')$, it suffices to define $e'(e^{-1}\ \star) = e\ \star$, with the rest of $e' : \mathsf{Fin}\ n_1' \simeq \mathsf{Fin}\ n_2'$ defined as a restriction of e. This construction corresponds more generally to the *Gordon complementary bijection principle* [Gordon, 1983], whereby a bijection $A_1 \xrightarrow{\sim} B_1$ can be constructively "subtracted" from a bijection $(A_0 + A_1) \xrightarrow{\sim} (B_0 + B_1)$, yielding a bijection $A_0 \xrightarrow{\sim} B_0$. (Unfortunately, I do not currently know of a good way to encode a proof of the fully general bijection principle in a constructive logic.) ∎

Remark. It seems somewhat strange that the above proof has so much computational content—the required manipulations of equivalences are quite nontrivial—when the end goal is to prove a mere proposition. I do not know whether there is a simpler proof.

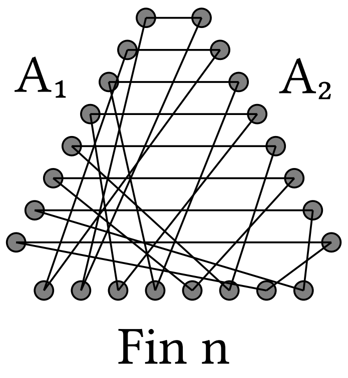


Figure 2.4: A path between inhabitants of $\mathcal{U}_{\mathrm{Fin}}$ contains only triangles

there are only two degrees of freedom. Once the evidence of finiteness is determined for A_1 and A_2, there is only one valid correspondence between them—but there ought to be $n!$ such correspondences.

Proposition 2.4.6. *$\mathcal{U}_{\mathrm{Fin}}$ is a set, that is, for any $X, Y : \mathcal{U}_{\mathrm{Fin}}$, if $p_1, p_2 : X = Y$ then $p_1 = p_2$.*

Proof (sketch). A path $(A_1, n_1, e_1) = (A_2, n_2, e_2)$ is equivalent to $(p : A_1 = A_2) \times (q : n_1 = n_2) \times (q_*(p_*(e_1)) = e_2)$. The transport of e_1 by p is given by the composition $e_1 \circ (\mathsf{ua}^{-1}(p))^{-1}$, but this essentially means that p is uniquely determined by e_1 and e_2. SDG

The underlying problem is that $\mathcal{U}_{\mathrm{Fin}}$ does not actually do a very good job at encoding what classical mathematicians usually mean by "finite set". Saying that a set A is finite with size n does not typically imply there is some specific, chosen bijection $A \xrightarrow{\sim} [n]$, but merely that A *can be* put in bijection with $[n]$, with no mention of a specific bijection. This is justified by the fact that, up to isomorphism, any bijection $A \xrightarrow{\sim} [n]$ is just as good as any other.

This suggests a better encoding of finiteness in type theory.

Definition 2.4.7. The type of finite sets is given by

$$\mathcal{U}_{\|\mathrm{Fin}\|} :\equiv (A : \mathcal{U}) \times \mathsf{isFinite}(A),$$

where

$$\mathsf{isFinite}(A) :\equiv \|(n : \mathbb{N}) \times (A \simeq \mathsf{Fin}\ n)\|.$$

Here we make use of propositional truncation to encode the fact that there *merely exists* some size n and an equivalence between A and $\mathsf{Fin}\ n$, but without exposing a precise choice. The finiteness evidence is now irrelevant to paths in $\mathcal{U}_{\|\mathrm{Fin}\|}$, since there is always a path between any two elements of a truncated type.

In an abuse of notation, we will often write $A : \mathcal{U}_{\|\mathrm{Fin}\|}$ instead of $(A, f) : \mathcal{U}_{\|\mathrm{Fin}\|}$ where $f : \mathsf{isFinite}(A)$. We first record a few properties of $\mathcal{U}_{\|\mathrm{Fin}\|}$.

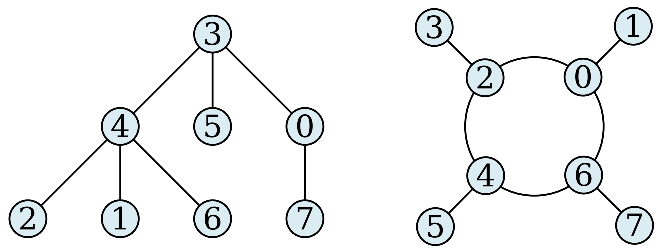


Figure 3.1: Representative labelled shapes

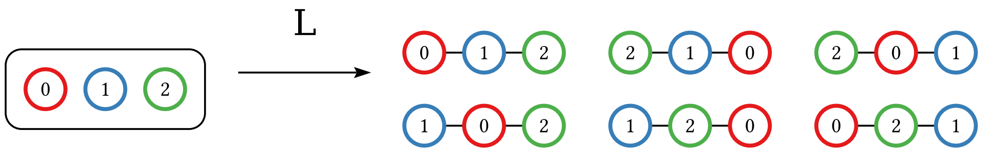


Figure 3.2: The species L of lists

More abstractly, *unlabelled* shapes can be defined as equivalence classes of labelled shapes (§3.3.2), which is nontrivial in the case of shapes with symmetry.

Besides its focus on labels, the power of the theory of species also derives from its ability to describe structures of interest *algebraically*, making them amenable to analysis with only a small set of general tools.

Example. Consider the species L of *lists*, or *linear orderings*; Figure 3.2 illustrates all the labelled list structures (containing each label exactly once) on the set of labels $[3] = \{0, 1, 2\}$. Of course, there are exactly $n!$ such list structures on any set of n labels.

The species of lists can be described by the recursive algebraic expression

$$\mathsf{L} = \mathsf{1} + \mathsf{X} \cdot \mathsf{L}.$$

The meaning of this will be made precise later. For now, its intuitive meaning should be clear to anyone familiar with recursive algebraic data types in a language such as Haskell or OCaml: a labelled list (L) is empty ($\mathsf{1}$) or ($+$) a single label (X) together with ($\cdot$) another labelled list (L).

Example. As another example, consider the species B of *(rooted, ordered) binary trees*. The set of all labelled binary trees on $\{0, 1, 2\}$ is shown in Figure 3.3.

Algebraically, such trees can be described by

$$\mathsf{B} = \mathsf{1} + \mathsf{B} \cdot \mathsf{X} \cdot \mathsf{B}.$$

Example. The species E of *sets* describes shapes consisting simply of an unordered collection of unique labels, with no other structure imposed. There is exactly one such shape for any set of labels, as illustrated in Figure 3.4.

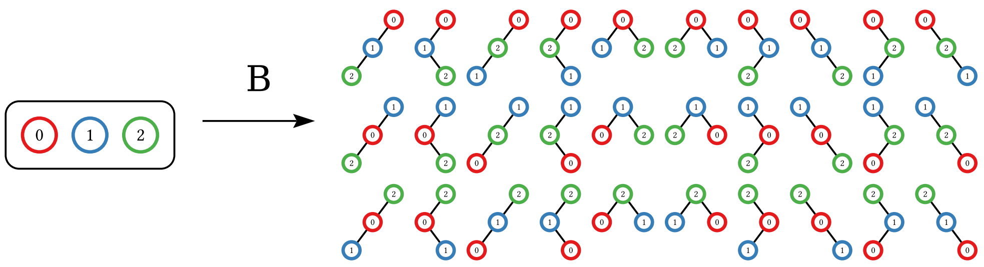


Figure 3.3: The species B of binary trees

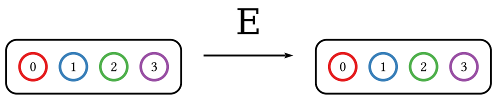


Figure 3.4: The species E of sets

Example. The species Mob of *mobiles* consists of non-empty binary trees where each node has exactly zero or two subtrees, and sibling subtrees are considered unordered. Figure 3.5 shows a single example Mob-shape, drawn in four (equivalent) ways. Algebraically,

$$\mathsf{Mob} = \mathsf{X} + \mathsf{X} \cdot (\mathsf{E}_2 \circ \mathsf{Mob}),$$

that is, a mobile is either a single label, or a label together with an unordered pair (E_2) of ($\circ$) mobiles.

Example. The species C of *cycles*, illustrated in Figure 3.6, describes shapes that consist of an ordered cycle of labels. One way to think of the species of cycles is as a quotient of the species of lists, where two lists are considered equivalent if one is a cyclic rotation of the other (see §4.6).

Example. The species S of *permutations*—*i.e.* bijective endofunctions—is illustrated in Figure 3.7. Algebraically, it can be described by

$$\mathsf{S} = \mathsf{E} \circ \mathsf{C},$$

that is, a permutation is a set of cycles.

Example. The species End of *endofunctions* consists of directed graphs corresponding to valid endofunctions on the labels—that is, where every label has exactly one outgoing edge (Figure 3.8).

Some reflection shows that endofunctions can be characterized as permutations of rooted trees,

$$\mathsf{End} = \mathsf{S} \circ T = \mathsf{E} \circ \mathsf{C} \circ T,$$

where $T = \mathsf{X} \cdot (\mathsf{E} \circ T)$. Each element which is part of a cycle serves as the root of a tree; iterating an endofunction starting from any element must eventually reach a

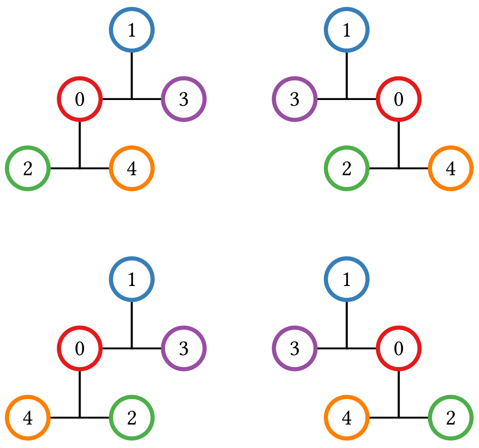


Figure 3.5: An example Mob-shape, drawn in four equivalent ways

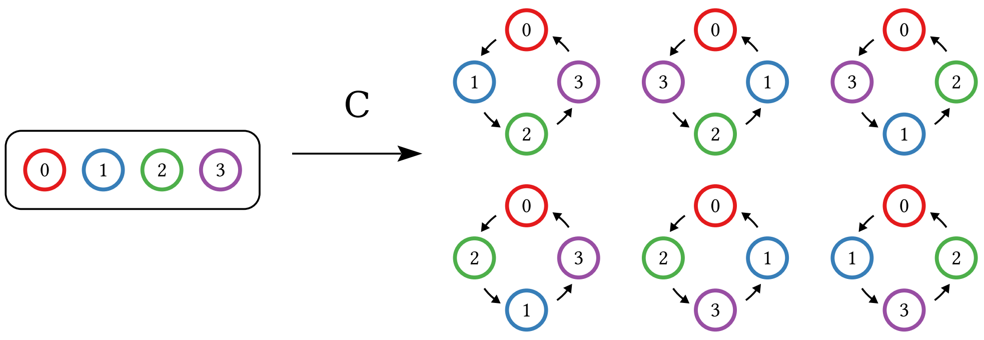


Figure 3.6: The species C of cycles

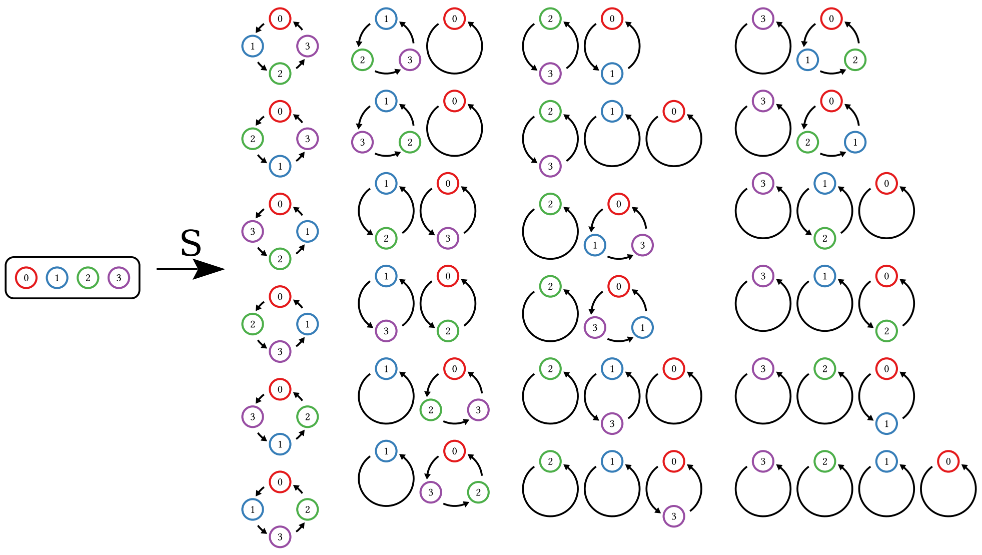


Figure 3.7: The species S of permutations

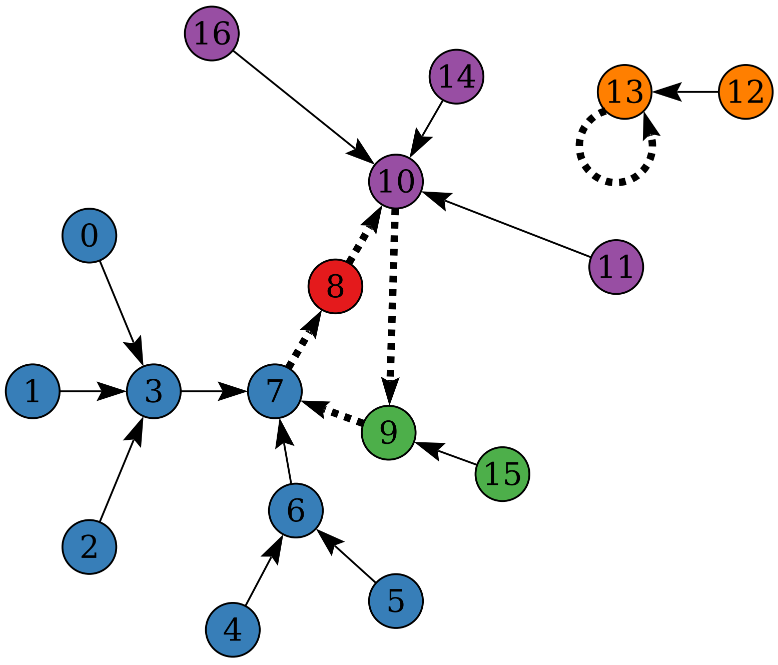


Figure 3.8: An example End-shape

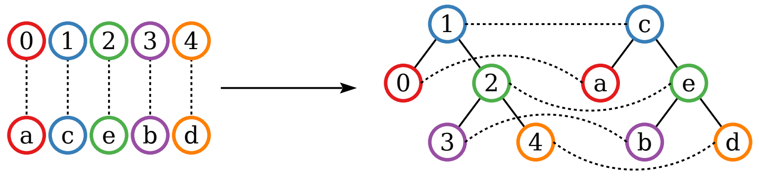


Figure 3.9: Relabelling

comfortable with category theory, it may be hard to grasp the intuition for the abstract definition right away.

Definition 3.2.2. A *species* F is a pair of mappings which

- sends any finite set L (of *labels*) to a set $F\ L$ (of *shapes*), and
- sends any bijection on finite sets $\sigma : L \xrightarrow{\sim} L'$ (a *relabelling*) to a function $F\ \sigma : F\ L \to F\ L'$ (illustrated in Figure 3.9),

satisfying the following functoriality conditions:

- $F\ id_L = id_{FL}$, and
- $F\ (\sigma \circ \tau) = F\ \sigma \circ F\ \tau$.

We call $F\ L$ the set of "F-shapes with labels drawn from L", or simply "F-shapes on L", or even (when L is clear from context) just "F-shapes".[1] $F\ \sigma$ is called the "transport of σ along F", or sometimes the "relabelling of F-shapes by σ".

The functoriality of a species F means that the actual labels used don't matter; the resulting family of shapes is independent of the particular labels used. We might say that species are *parametric* in label sets of a given size. In particular, F's action on all label sets of size n is determined by its action on any particular such set: if $|L_1| = |L_2|$ and we know $F\ L_1$, we can determine $F\ L_2$ by lifting an arbitrary bijection between L_1 and L_2. More formally, although Definitions 3.2.1 and 3.2.2 say only that a species F sends a bijection $\sigma : L \xrightarrow{\sim} L'$ to a *function* $F\ \sigma : F\ L \to F\ L'$, the functoriality of F guarantees that $F\ \sigma$ is a bijection as well. In particular, $(F\ \sigma)^{-1} = F\ (\sigma^{-1})$, since $F\ \sigma \circ F\ (\sigma^{-1}) = F\ (\sigma \circ \sigma^{-1}) = F\ id = id$, and similarly $F\ (\sigma^{-1}) \circ F\ \sigma = id$. Thus, *up to isomorphism*, a functor F must do the same thing for any two label sets of the same size.

We may therefore take the finite set of natural numbers $[n] = \{0, \ldots, n-1\}$ as *the* canonical label set of size n, and write $F\ n$ (instead of $F\ [n]$) for the set of F-shapes

[1]Margaret Readdy's translation of Bergeron et al. [1998] uses the word "structure" instead of "shape", but that word is likely to remind computer scientists of "data structures", which is, again, the wrong association: data structures contain *data*, whereas species shapes contain only labels. I try to consistently use the word "shape" to refer to the elements of a species, and reserve "structure" for the labelled data structures to be introduced in Chapter 6.

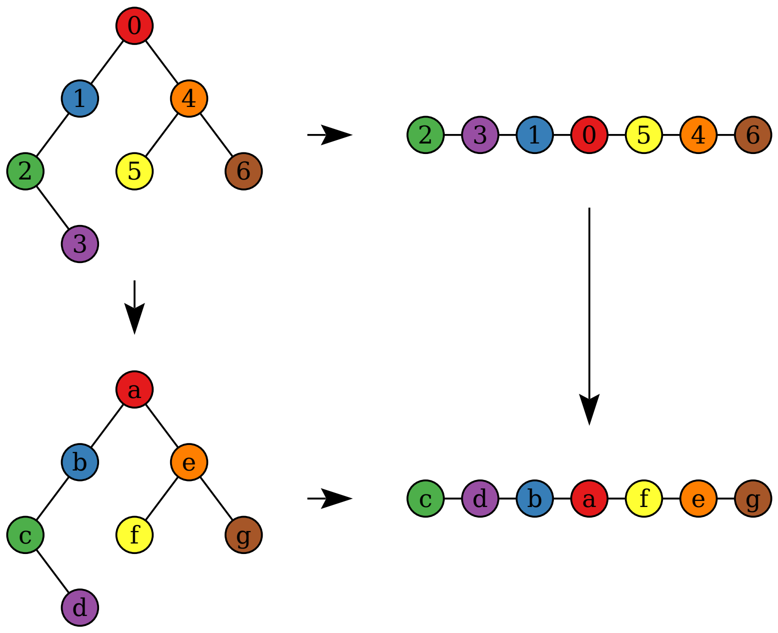


Figure 3.10: Inorder traversal is natural

where

$$\mathsf{isFinite}(A) :\equiv \|(n : \mathbb{N}) \times (A \simeq \mathsf{Fin}\ n)\|$$

and with morphisms given by paths.

Definition 3.2.3. A *constructive species*, or h-species, is an h-functor $F : \mathcal{B} \to \mathcal{S}$. We use $\mathbf{Spe} = \mathcal{B} \Rightarrow \mathcal{S}$ to refer to the h-category of constructive species, the same name as the category $\mathbf{B} \Rightarrow \mathbf{Set}$ of set-theoretic species; while technically ambiguous, this should not cause confusion since it should always be clear from the context whether we are working in set theory or in HoTT. Likewise, when working in the context of HoTT we will often simply say "species" instead of "constructive species".

The above definition corresponds directly to the definition of species in set theory. However, it is more specific than necessary. In fact, in HoTT, *any* function of type $\mathcal{B} \to \mathcal{S}$ (that is, a function from objects of $\mathcal{B}$ to objects of $\mathcal{S}$) is automatically an h-functor. Since the morphisms in $\mathcal{B}$ are just paths, functoriality corresponds to transport. Thus, as hinted in Chapter 1, within HoTT it is simply impossible to write down an invalid species. This is a strong argument for working within type theory in general and HoTT in particular: it provides exactly the right sort of type system which allows expressing only valid species.

Nevertheless, it is still not perfectly clear whether this is the right encoding of species within homotopy type theory. It cannot be directly justified by showing that $\mathbf{B} \Rightarrow \mathbf{Set}$ and $\mathcal{B} \Rightarrow \mathcal{S}$ are categorically equivalent; this does not even make sense since they live in entirely different foundational frameworks. Rather, a justification

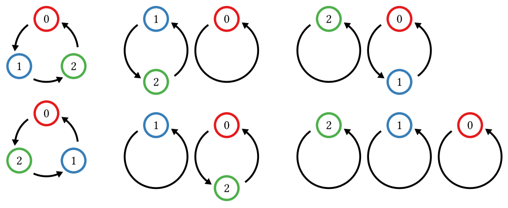


Figure 3.11: Permutations of size three

must be extensional, in the sense of showing that the two definitions have similar properties and support similar operations. In a sense, much of Chapter 4 is precisely such an extensional justification.

3.3 Isomorphism and equipotence

Just as with HoTT itself, *sameness* and related notions are also at the heart of the theory of species. In this section we explore isomorphism of species and of species shapes, as well as a coarser notion of equivalence on species known as *equipotence*.

3.3.1 Species isomorphism

An isomorphism of species is just an isomorphism in the category of species, that is, a pair of inverse natural transformations. Species isomorphism preserves all the interesting *combinatorial* properties of species; hence in the combinatorics literature everything is always done up to isomorphism. However, this is usually done in a way that glosses over the *computational* properties of the isomorphisms. Formulating species within HoTT gives us the best of both worlds: naturally isomorphic functors between h-categories are equal, and hence isomorphic species are literally identified; however, equalities (*i.e.* paths) in HoTT may still have computational content.

3.3.2 Shape isomorphism and unlabelled species

In addition to isomorphism of entire species, there is also a natural notion of isomorphism for individual species *shapes*. For example, consider the set of permutations on the labels $\{0, 1, 2\}$, shown in Figure 3.11. Notice that some of these permutations "have the same form". For example, the only difference between the two permutations shown in Figure 3.12 is their differing labels. On the other hand, the two permutations shown in Figure 3.13 are fundamentally different, in the sense that there is no way to merely *relabel* one to get the other.

We can formalize this idea as follows.

Definition 3.3.1. Given a species F and F-shapes $f_1 : F\ L_1$ and $f_2 : F\ L_2$, we say f_1 and f_2 are *equivalent up to relabelling*, or *have the same form*, and write $f_1 \approx f_2$, if

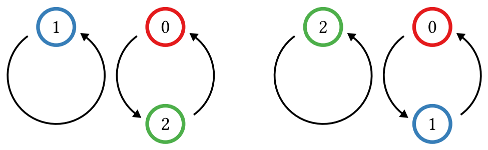


Figure 3.12: Two permutations with the same form

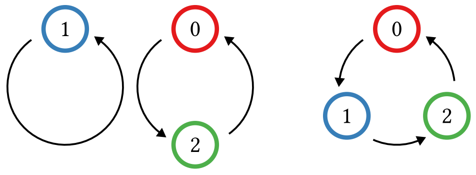


Figure 3.13: Two permutations with different forms

there is some bijection $\sigma : L_1 \xrightarrow{\sim} L_2$ such that $F\ \sigma\ f_1 = f_2$. If we wish to emphasize the particular bijection relating f_1 and f_2 we may write $f_1 \approx_\sigma f_2$.

Thus, the two labelled shapes shown in Figure 3.12 are related by $\approx$, whereas those shown in Figure 3.13 are not.

Definition 3.3.2. Given a species F, denote by $\mathrm{sh}(F)$ the groupoid whose objects are F-shapes—that is, finite sets L together with an element of $F\ L$—and whose morphisms are given by the $\approx$ relation.

Proof. We need to show this is a well-defined groupoid, *i.e.* that $\approx$ is an equivalence relation. The $\approx$ relation is reflexive, yielding identity morphisms, since any shape is related to itself by the identity bijection. If $f \approx g \approx h$ then $f \approx h$ by composing the underlying bijections. Finally, $f \approx g$ implies $g \approx f$ since the underlying bijections are invertible. SDG

Given these preliminary definitions, we can now define what we mean by a *form*, or *unlabelled shape*.

Definition 3.3.3. An F-*form* is an equivalence class under $\approx$, that is, a connected component of the groupoid $\mathrm{sh}(F)$.

In other words, an F-form is a maximal class of labelled F-shapes which are all interconvertible by relabelling, that is, a maximal clique. As defined, such classes are rather large, as they include labellings by *all possible* sets of labels! Typically, we consider only a single label set of each size, such as $\mathsf{Fin}\ n$. For example, Figure 3.14 shows all the S-forms of size four, using two different representations: on the right

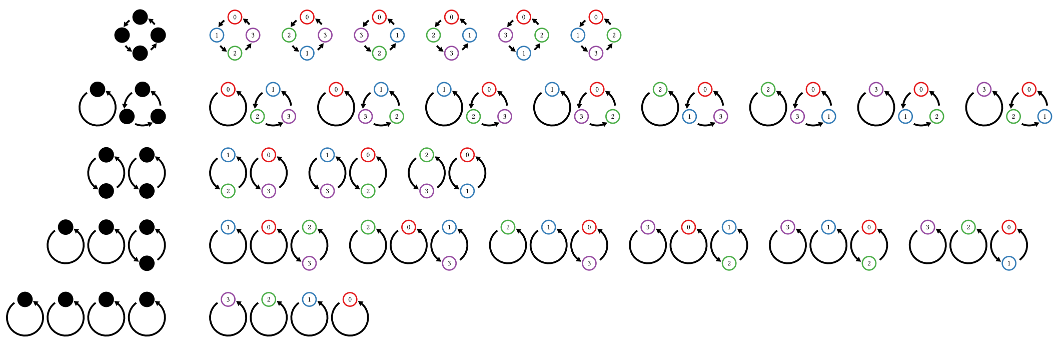


Figure 3.14: S-forms of size 4

are the literal equivalence classes of permutations on Fin 4 which are equivalent up to relabelling. On the left are schematic representations of each form, drawn by replacing labels with indistinguishable dots. Note that the schematic representations, while convenient, can break down in more complex situations, so it is important to also keep in mind the underlying definition in terms of equivalence classes.

Remark. What are here called *forms* are more often called *types* in the species literature; but using that term would lead to unnecessary confusion in the present context.

3.3.3 Equipotence

It turns out that there is another useful equivalence relation on species which is *weaker* (*i.e.* coarser) than isomorphism/equality, known as *equipotence.*

Definition 3.3.4. An *equipotence* between species F and G, denoted $F \overset{\#}{=} G$,[2] is defined as an "unnatural" isomorphism between F and G—that is, two families of functions $\varphi_L : F\ L \to G\ L$ and $\psi_L : G\ L \to F\ L$ such that $\varphi_L \circ \psi_L = \psi_L \circ \varphi_L = id$ for every finite set L. Note in particular there is no requirement that φ or ψ be natural.

We can see that an equipotence preserves the *number* of shapes of each size, since φ and ψ constitute a bijection, for each label set L, between the set of F-shapes $F\ L$ and the set of G-shapes $G\ L$. Isomorphic species are of course equipotent, where the equipotence also happens to be natural. It may be initially surprising, however, that the converse is false: there exist equipotent species which are not isomorphic. Put another way, having the same number of structures of each size is not enough to ensure isomorphism.

One good example is the species L of lists and the species S of permutations. As is well-known, there are the same number of linear orderings of n labels as there are permutations of n labels (namely, $n!$). In fact, this is so well-known that mathematicians

[2]In the species literature, equipotence is usually denoted $F \equiv G$, but we are already using that symbol to denote judgmental equality.

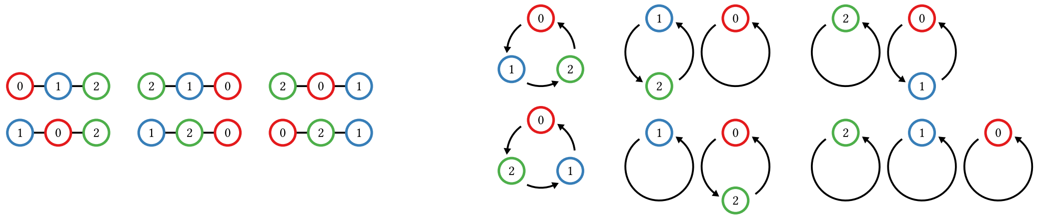


Figure 3.15: Lists and permutations on three labels

routinely conflate the two, referring to an ordered list as a "permutation". Figure 3.15 shows the six lists and six permutations on three labels.

However, L and S are not isomorphic. The intuitive way to see this is to note that although there is only a single list form of any given size, for $n \geqslant 2$ there are multiple permutation forms. Every permutation, *i.e.* bijective endofunction, can be decomposed into a set of cycles, and a relabelling can only map between permutations with the same number of cycles of the same sizes. There is thus one S-form corresponding to each integer partition of n (Figure 3.14 shows the five permutation forms of size 4, corresponding to $4 = 3 + 1 = 2 + 2 = 2 + 1 + 1 = 1 + 1 + 1 + 1$).

More formally, suppose there were some *natural* isomorphism witnessed by $\varphi : \mathsf{L} \overset{\bullet}{\longrightarrow} \mathsf{S}$ and $\psi : \mathsf{S} \overset{\bullet}{\longrightarrow} \mathsf{L}$. In particular, for any $\sigma : K \overset{\sim}{\longrightarrow} K$ we would then have

$$\begin{tikzcd} \mathsf{L}\ K \arrow[r, "\varphi_K"] \arrow[d, "\mathsf{L}\ \sigma"'] & \mathsf{S}\ K \arrow[d, "\mathsf{S}\ \sigma"] \\ \mathsf{L}\ K \arrow[r, "\varphi_K"'] & \mathsf{S}\ K \end{tikzcd}$$

and similarly for ψ_K in the opposite direction. This says that any two K-labelled lists related by the relabelling σ correspond to permutations which are also related by σ. However, as we have seen, *any* two lists are related by some relabelling, and thus (since φ and ψ constitute a bijection) any two permutations would have to be related by some relabelling as well, but this is false.

This argument shows that there cannot exist a natural isomorphism between L and S. However, the claim is that they are nonetheless equipotent. Again, this fact is very well known, but it is still instructive to work out the details of a formal proof.

The first and most obvious "proof" is to send the permutation $\sigma : (\mathsf{Fin}\ n)!$ to the list whose ith element is $\sigma(i)$, and vice versa. Note, however, that this is not really a proof, since it only gives us a specific bijection $\mathsf{L}\ (\mathsf{Fin}\ n) \overset{\sim}{\longrightarrow} \mathsf{S}\ (\mathsf{Fin}\ n)$, rather than a family of bijections $\mathsf{L}\ K \overset{\sim}{\longrightarrow} \mathsf{S}\ K$. We will return to this point shortly.

The second proof, known as the *fundamental transform*, is more elegant from a combinatorial point of view. For more details, see Cartier and Foata [1969], Knuth [1973], or Bergeron et al. [1998, p. 22]. We first describe the mapping from permutations on $\mathsf{Fin}\ n$ to lists on $\mathsf{Fin}\ n$: given a permutation, order its cycles in decreasing order

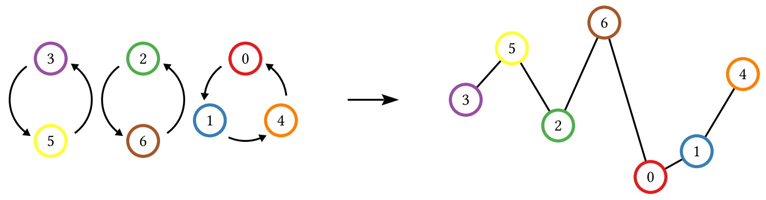


Figure 3.16: The fundamental transform

of their smallest element, and then transcribe each cycle as a list beginning with the smallest element. Figure 3.16 shows an example where the permutation (35)(26)(014) (whose cycles have minimum elements 3, 2, and 0 respectively) is sent to the list 3526014, which for emphasis is drawn with the height of each node corresponding to the size of its label. To invert the transformation, partition a list into segments with each record minimum beginning a new segment, and turn each such segment into a cycle. For example, in the list 3526014, the elements 3, 2, and 0 are the ones which are smaller than all the elements to their left, so each one marks off the beginning of a new cycle.

The way the fundamental transform is presented also makes it clear how to generalize from $\mathsf{Fin}\ n$ to other finite sets of labels L: all we require is a linear order on L, in order to find the minimum label in a given cycle and sort the cycles by minimum element, and to determine the successive record minima in a list. Looking back at the first, "obvious" proof, which sends σ to the list whose ith element is $\sigma(i)$, we can see that it also can be generalized to work for any finite set L equipped with a linear order. In particular, being equipped with a linear order is equivalent to being equipped with a bijection to $\mathsf{Fin}\ n$, as explained in §2.4.3.

Intuitively, then, the reason that these two families of bijections are not natural is that they do not work *uniformly* for all sets of labels, but require some extra structure. Any finite label set can be given a linear order, but the precise choice of linear order determines how the bijections work.

Considering this from the viewpoint of HoTT yields additional insight. A family of functions like φ_K would typically correspond in HoTT to a function of type

$$\varphi : (K : \mathcal{U}_{\|\mathrm{Fin}\|}) \to \mathsf{L}\ K \to \mathsf{S}\ K.$$

It is certainly possible to implement a function with the above type (for example, one which sends each list to the cyclic permutation with elements in the same order), but as we have seen, it is not possible to implement one which is invertible. Writing an invertible such function also requires a linear ordering on the type K. We could, of course, simply take a linear order as an extra argument,

$$\varphi : (K : \mathcal{U}_{\|\mathrm{Fin}\|}) \to \mathsf{LinOrd}\ K \to \mathsf{L}\ K \to \mathsf{S}\ K.$$

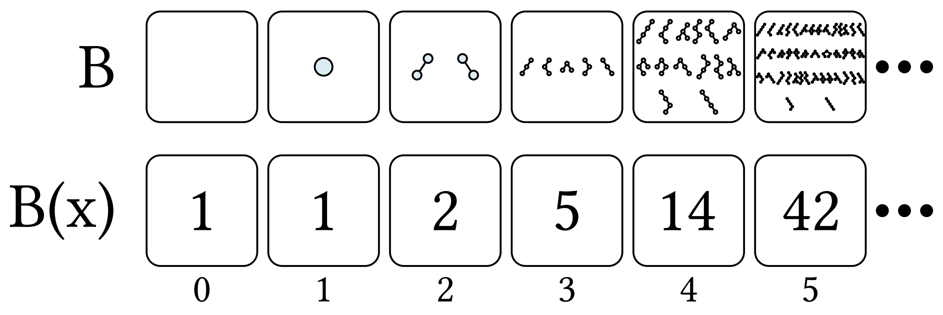


Figure 3.17: Correspondence between species and generating functions

One can see that the mapping from species to generating functions discards information, compressing an entire set of shapes or forms into a single number (Figure 3.17). Once one has defined the notion of species, it is not hard to come up with the notion of generating functions as a sort of "structured summary" of species.

Historically, however, generating functions came first. As Joyal makes explicit in the introduction to his seminal paper *Une Théorie Combinatoire des Séries Formelles* [1981]—in fact, it is even made explicit in the title of the paper itself—the main motivation for inventing species was to generalize the theory of generating functions, putting it on firmer combinatorial and categorical ground. The theory of generating functions itself was already well-developed, but no one had yet tried to view it through a categorical lens.

The general idea is to "blow everything up", replacing natural numbers by sets; addition by disjoint union; product by pairing; and so on. In a way, one can see this process as "imbuing everything with constructive significance"; this is one argument for the naturalness of developing the theory of species within a constructive type theory.

3.5 Conclusion

In this chapter we have seen the definition of species, both in set theory and type theory, and related definitions such as isomorphism and equipotence of species and generating functions. We have seen that defining species within homotopy type theory has some benefits: for example, it becomes impossible to write down invalid species within the type theory, and homotopy type theory sheds new light on some of the fundamental equivalence relations on species. However, up to this point everything has been "low-level", in the sense of working directly with the definition of species. In the next chapter we will see how to build a higher-level algebraic framework on top of species, and how this also gives us a framework for generalizing species to other categories.

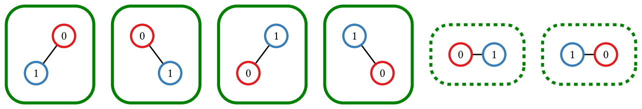


Figure 4.1: $(\mathsf{B} + \mathsf{L})\ 2$

$$
\begin{aligned}
& (F + G)(f \circ g) \\
= & \qquad\qquad\qquad \{ \quad + \text{ definition} \quad \} \\
& F\ (f \circ g) \uplus G\ (f \circ g) \\
= & \qquad\qquad\qquad \{ \quad F, G \text{ functors} \quad \} \\
& (F\ f \circ F\ g) \uplus (G\ f \circ G\ g) \\
= & \qquad\qquad\qquad \{ \quad \uplus \text{ bifunctor} \quad \} \\
& (F\ f \uplus G\ f) \circ (F\ g \uplus G\ g) \\
= & \qquad\qquad\qquad \{ \quad + \text{ definition} \quad \} \\
& (F + G)\ f \circ (F + G)\ g.
\end{aligned}
$$

SDG

Remark. More abstractly, when defining a functor with a groupoid as its domain (such as $F + G$ above), it suffices to specify only its action on objects, using an arbitrary expression composed of (co- and contravariant) functors. For example, $(F + G)\ L = F\ L \uplus G\ L$ is defined in terms of the functors F, G, and $\uplus$. In that case the action of the functor on morphisms can be derived automatically by induction on the structure of the expression, simply substituting the morphism in place of covariant occurrences of the object, and the morphism's inverse in place of contravariant occurrences. In fact, in HoTT, this is simply transport; that is, given an h-groupoid B and a (pre)category C, any function $B_0 \to C_0$ extends to a functor $B \to C$.

By the same token, to define a functor with an arbitrary category (not necessarily a groupoid) as its domain, it suffices to define its action on an object using an expression containing only covariant occurrences of the object.

Example. $\mathsf{B} + \mathsf{L}$ is the species of shapes which are *either* binary trees or lists (Figure 4.1).

Example. As another example, consider $\mathsf{B} + \mathsf{B}$. It is important to bear in mind that $+$ yields a *disjoint* or "tagged" union; so $\mathsf{B} + \mathsf{B}$ consists of *two* copies of every binary tree (Figure 4.2), and in particular it is distinct from B.

Species sum corresponds to the sum of generating functions: we have

$$(F + G)(x) = F(x) + G(x) \quad \text{and} \quad \widetilde{(F + G)}(x) = \widetilde{F}(x) + \widetilde{G}(x).$$

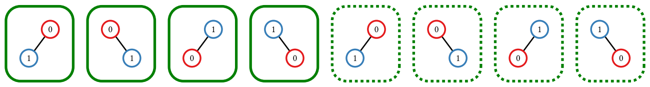


Figure 4.2: $(\mathsf{B} + \mathsf{B})\ 2$

This is because the sum of two generating functions is computed by summing corresponding coefficients,

$$\left(\sum_{n\geqslant 0} a_n x^n\right) + \left(\sum_{n\geqslant 0} b_n x^n\right) = \sum_{n\geqslant 0}(a_n + b_n)x^n$$

(and likewise for egfs), and since species sum is given by disjoint union, the number of $(F + G)$-shapes and -forms of a given size is the sum of the number of F- and G-shapes (respectively -forms) of that size.

There is also a primitive species which is an identity element for species sum.

Definition 4.1.2. The *zero* or *empty* species, $\mathsf{0}$, is the unique species with no shapes whatsoever. That is, on objects, $\mathsf{0}\ L := \varnothing$, and on morphisms $\mathsf{0}$ sends every σ to the unique function $\varnothing \to \varnothing$.

We evidently have

$$\mathsf{0}(x) = \widetilde{\mathsf{0}}(x) = 0 + 0x + 0x^2 + \cdots = 0.$$

Proposition 4.1.3. $(+, \mathsf{0})$ *is a symmetric monoid on* $\mathbf{B} \Rightarrow \mathbf{Set}$.

Proof. First, we must show that $+$ is a bifunctor. By definition it sends two functors to a functor, but this is only its action on the objects of $\mathbf{Spe}$. We must also specify its action on morphisms, that is, natural transformations between species, and we must show that it preserves identity natural transformations and (vertical) composition of natural transformations.

In this case it's enough simply to unfold definitions and follow the types. Given species F, F', G, and G' and natural transformations $\phi : F \xrightarrow{\bullet} F'$ and $\psi : G \xrightarrow{\bullet} G'$, we should have $\phi + \psi : F + G \xrightarrow{\bullet} F' + G'$. The component of $\phi + \psi$ at some $L \in \mathbf{B}$ should thus be a morphism in $\mathbf{Set}$ of type $F\ L \uplus G\ L \to F'\ L \uplus G'\ L$; the only thing that fits the bill is $\phi_L \uplus \psi_L$.

This nicely fits with the "elementwise" definition of $+$ on species: $(F + G)\ L = F\ L \uplus G\ L$, and likewise $(\phi + \psi)_L = \phi_L \uplus \psi_L$. The action of $+$ on natural transformations thus reduces to the elementwise action of $\uplus$ on their components. From this it follows that

- $\phi + \psi$ is natural (because ϕ and ψ are), and
- $+$ preserves identity and composition (because $\uplus$ does).

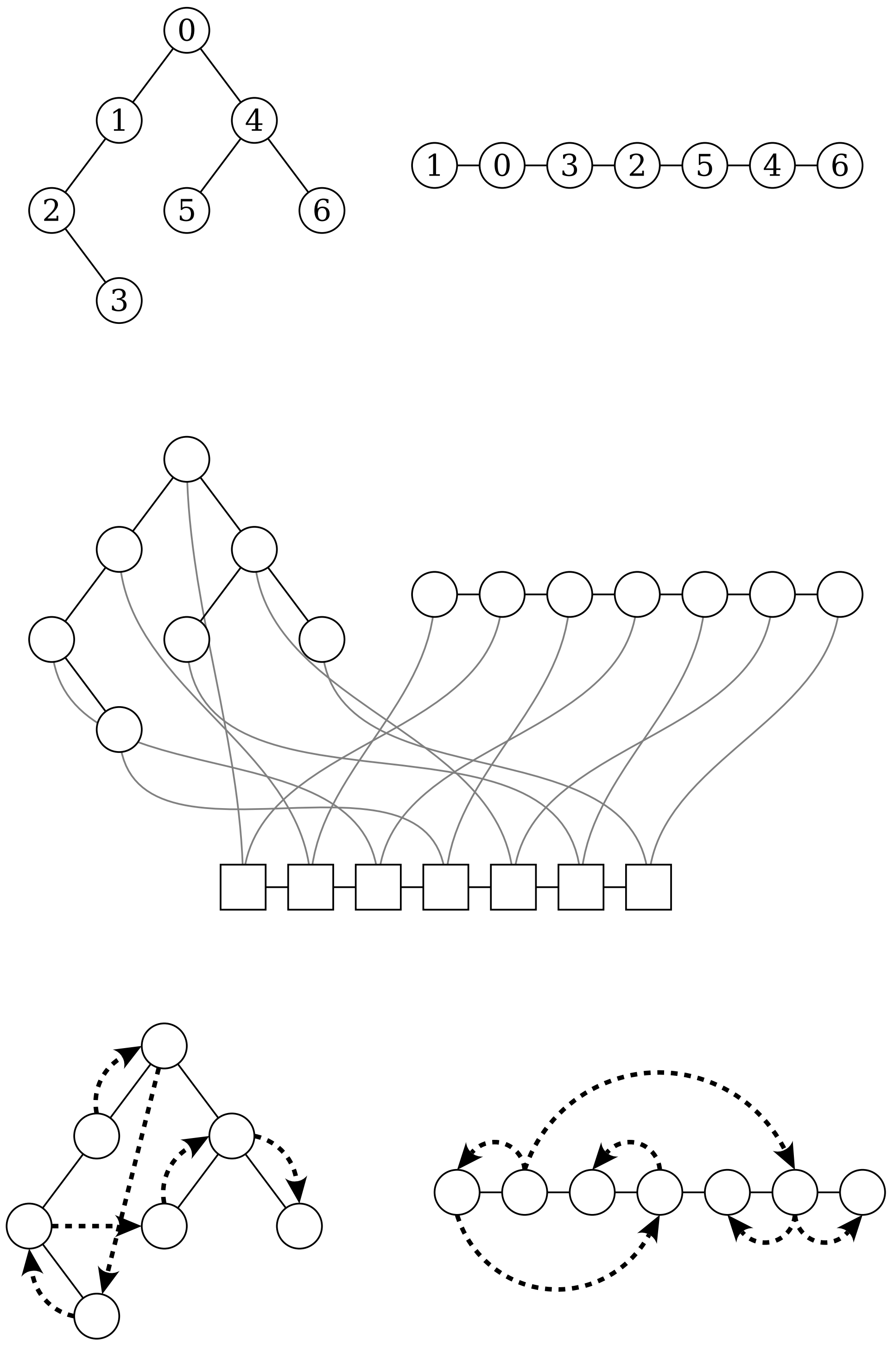


Figure 4.3: Four views on the Cartesian product $\mathsf{B} \times \mathsf{L}$

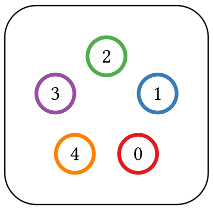


Figure 4.4: The unique $\mathsf{E}\ 5$ shape

and

$$\left(\sum_{n\geqslant 0} a_n \frac{x^n}{n!}\right) \times \left(\sum_{n\geqslant 0} b_n \frac{x^n}{n!}\right) = \sum_{n\geqslant 0} (a_n b_n) \frac{x^n}{n!}$$

denote the elementwise or *Hadamard* product of two generating functions. This is not a particularly natural operation on generating functions (although it is easy to compute); in particular it is not what one usually thinks of as *the* product of generating functions. As we will see in §4.2, there is a different combinatorial operation that corresponds to the usual product of generating functions.

There is also a species, usually called E, which is an identity element for Cartesian product. Considering that we should have $(\mathsf{E} \times G)\ L = \mathsf{E}\ L \times G\ L \simeq G\ L$, the proper definition of E becomes clear:

Definition 4.1.5. The species of *sets*, E, is defined as the constant functor yielding $\{\star\}$, that is, $\mathsf{E}\ L = \{\star\}$.

The ogf for E is given by

$$\widetilde{\mathsf{E}}(x) = 1 + x + x^2 + \cdots = \frac{1}{1-x},$$

and the egf by

$$\mathsf{E}(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = e^x.$$

The notation E was probably chosen as an abbreviation of the French *ensemble* (set), but it is also a clever pun on the fact that $\mathsf{E}(x) = e^x$.

Remark. E is called the *species of sets* since there is exactly one shape on any set of labels, which can be thought of as the set of labels itself, with no additional structure. In fact, since all one-element sets are isomorphic, we may define $\mathsf{E}\ L = \{L\}$ (Figure 4.4).

Proposition 4.1.6. $(\times, \mathsf{E})$ *is a symmetric monoid on* **Spe**.

Proof. The proof is omitted, since it is almost exactly the same as the proof for $(+, \mathsf{0})$; the only difference is the substitution of Cartesian product of sets for disjoint union. □

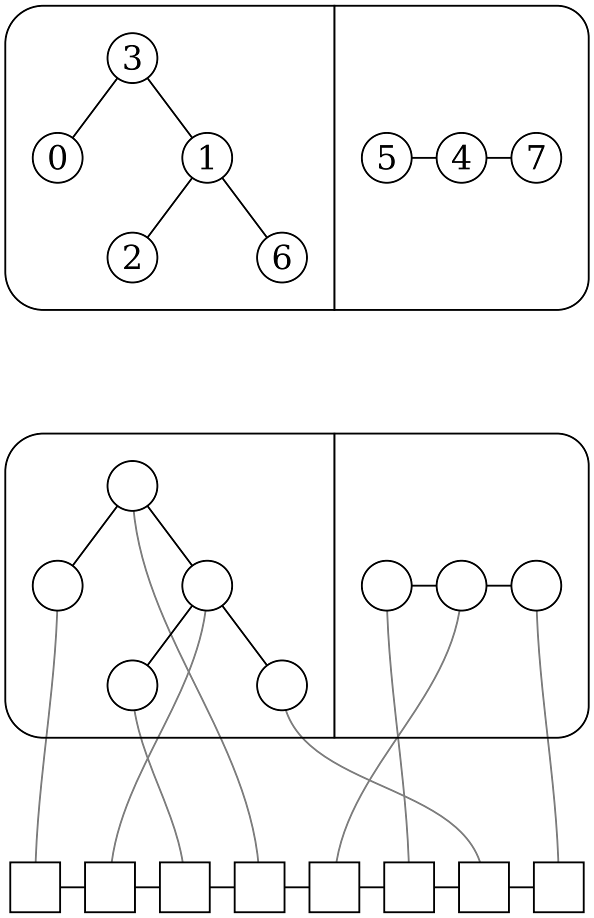


Figure 4.5: Two views on the partitional species product $\mathsf{B} \cdot \mathsf{L}$

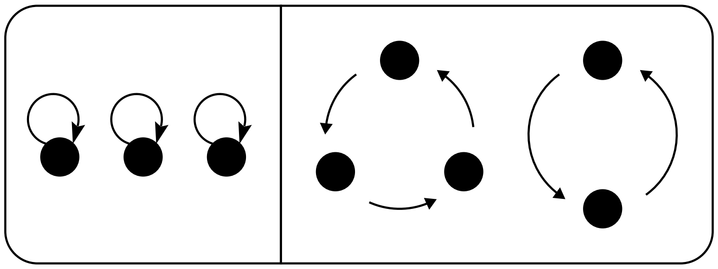


Figure 4.6: Permutation = fixpoints · derangement

Definition 4.2.2. The *unit species*, $\mathsf{1}$, is defined by

$$\mathsf{1}\ L = \begin{cases} \{\star\} & L = \varnothing \\ \varnothing & \text{otherwise.} \end{cases}$$

Remark. Recall that one should not think of $\mathsf{1}$ as doing case analysis. Rather, a more intuitive way to think of it is "there is a single $\mathsf{1}$-shape, and it has no labels"; that is, the unit species denotes a sort of "trivial" or "leaf" structure containing no labels. Intuitively, it corresponds to a Haskell type like

```
data Unit a = Unit
```

The generating functions for $\mathsf{1}$ are given by

$$\mathsf{1}(x) = \widetilde{\mathsf{1}}(x) = 1.$$

Example. The following example is due to Joyal [1981]. Recall that S denotes the species of permutations. Consider the species Der of *derangements*, that is, permutations which have no fixed points. It is not possible, in general, to directly express species using a "filter" operation, as in, "all F-shapes satisfying predicate P". However, it is possible to get a handle on Der in a more constructive manner by noting that every permutation can be canonically decomposed as a set of fixed points paired with a derangement on the rest of the elements (Figure 4.6). That is, algebraically,

$$\mathsf{S} = \mathsf{E} \cdot \mathsf{Der}. \tag{4.2.1}$$

This does not directly give us an expression for Der, since there is no notion of multiplicative inverse for species[2]. However, this is still a useful characterization of derangements. For example, since the mapping from species to egfs is a homomorphism

[2]Multiplicative inverses can in fact be defined for suitable *virtual* species [Bergeron et al., 1998, Chapter 3]. However, virtual species are beyond the scope of this dissertation.

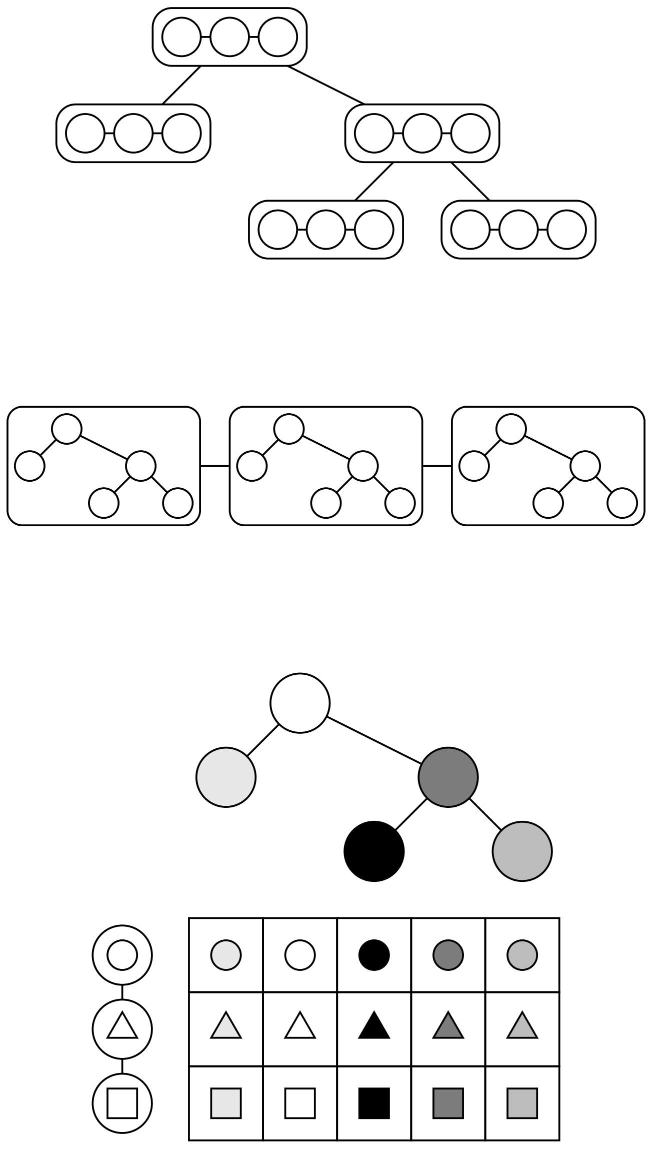


Figure 4.7: Three views on the arithmetic product $\mathsf{B} \boxtimes \mathsf{L}$

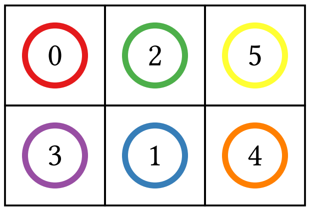


Figure 4.8: A Mat-shape of size 6

- $|X \cap Y| = 1$, for all $X \in \pi$, $Y \in \tau$.

Here, $\pi \vdash L$ denotes that π is a partition of L into any number of nonempty parts, that is, the elements of π are nonempty, pairwise disjoint, and have L as their union. We write $\pi, \tau \Vdash L$ to denote that (π, τ) constitute a rectangle on L, and call π and τ the *sides* of the rectangle.

We can now formally define arithmetic product as follows:

Definition 4.2.5. The *arithmetic product* $F \boxtimes G$ of two species F and G is the species defined on objects by

$$(F \boxtimes G)\ L = \biguplus_{L_F, L_G \Vdash L} F\ L_F \times G\ L_G.$$

$(F \boxtimes G)$ lifts bijections $\sigma : L \xrightarrow{\sim} L'$ to functions $(F \boxtimes G)\ L \to (F \boxtimes G)\ L'$ as follows:

$$(F \boxtimes G)\ \sigma\ (L_F, L_G, f, g) = (\mathcal{P}(\sigma)\ L_F, \mathcal{P}(\sigma)\ L_G, F\ \mathcal{P}(\sigma)\ f, G\ \mathcal{P}(\sigma)\ g),$$

where $\mathcal{P}(\sigma) : \mathcal{P}(L) \xrightarrow{\sim} \mathcal{P}(L')$ denotes the functorial lifting of σ to a bijection between subsets of L and L'.

Remark. The similarity of this definition to the definition of partitional product should be apparent: the only real difference is that rectangles $(L_F, L_G \Vdash L)$ have been substituted for partitions $(L_F, L_G \vdash L)$.

Example. $\mathsf{Mat} = \mathsf{L} \boxtimes \mathsf{L}$ is the species of (two-dimensional) *matrices*. Mat-shapes consist simply of labels arranged in a rectangular grid (Figure 4.8).

Example. $\mathsf{Rect} = \mathsf{E} \boxtimes \mathsf{E}$ is the species of *rectangles*. One way to think of rectangles is as equivalence classes of matrices up to reordering of the rows and columns. Each label has no fixed "position"; the only invariants on any given label are the sets of other

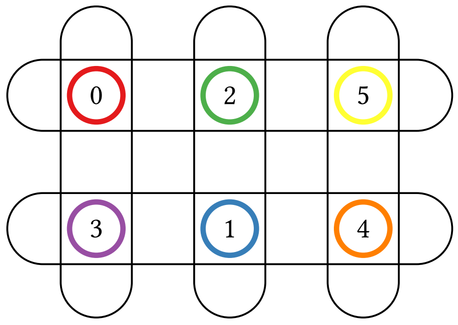


Figure 4.9: A Rect-shape of size 6

labels which are in the same row or column. Figure 4.9 shows an illustration; each rounded outline represents a *set* of labels. One can also take the species of rectangles as primitive and define arithmetic product in terms of it.

Example. Just as topological cylinders and tori may be obtained by gluing the edges of a square, species corresponding to cylinders or tori may be obtained by starting with the species of 2D matrices and "gluing" along one or both edges by turning lists L into cycles C. In particular, $\mathsf{Cyl} = \mathsf{L} \boxtimes \mathsf{C}$ is the species of (oriented) *cylinders*, and $\mathsf{Tor} = \mathsf{C} \boxtimes \mathsf{C}$ is the species of (oriented) *tori*.

Although species corresponding to Klein bottles and real projective planes (which arise from gluing the edges of a square with one or both pairs of edges given a half-twist before gluing, respectively) certainly exist, it does not seem they can be constructed using $\boxtimes$, since in those cases the actions of the symmetric group along the two axes are not independent.

The ogf for $F \boxtimes G$ is given by

$$\widetilde{F}(x) \boxtimes \widetilde{G}(x) = \left(\sum_{n \geqslant 0} f_n x^n \right) \boxtimes \left(\sum_{n \geqslant 0} g_n x^n \right) = \sum_{n \geqslant 0} \left(\sum_{d \mid n} f_d g_{n/d} \right) x^n,$$

since an $(F \boxtimes G)$-form of size n consists of a pair of an F-form and a G-form, whose sizes have a product of n.

Likewise, the egf is

$$\sum_{n \geqslant 0} \left(\sum_{d \mid n} \left\{ {n \atop d} \right\} f_d g_{n/d} \right) \frac{x^n}{n!},$$

where

$$\left\{ {n \atop d} \right\} = \frac{n!}{d!(n/d)!}$$

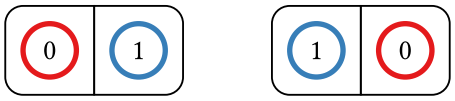


Figure 4.10: $(\mathsf{X} \cdot \mathsf{X})$-shapes

Example. Recall that L denotes the species of lists, *i.e.* linear orderings. Besides the interpretation of recursion, to be explored in §5.4.1, we have now seen all the necessary pieces to understand the algebraic definition of L:

$$\mathsf{L} = 1 + \mathsf{X} \cdot \mathsf{L}.$$

That is, a list structure is either the trivial structure on zero labels, or a single label paired with a list structure on the remainder of the labels. We also have $\mathsf{L} = 1 + \mathsf{X} + \mathsf{X}^2 + \mathsf{X}^3 + \dots$.

Example. Similarly, recall that the species B of *binary trees* is given by

$$\mathsf{B} = 1 + \mathsf{B} \cdot \mathsf{X} \cdot \mathsf{B}.$$

Example. The species $\mathsf{X} \cdot \mathsf{E}$ is variously known as the species of *pointed sets* (which may be denoted $\mathsf{E}^{\bullet}$) or the species of *elements* (denoted ε). $(\mathsf{X} \cdot \mathsf{E})$-structures consist of a single distinguished label paired with an unstructured collection of any number of remaining labels. There are thus n such structures on each label set of cardinality n, one for each label.

The two different names result from the fact that we may "care about" the labels in an E-structure or not—that is, when considering data structures built on top of species, E may correspond either to a bag data structure, or instead to a "sink" where we throw labels to which we do not wish to associate any data. This makes no difference from a purely combinatorial point of view, but makes a difference when considering labelled structures (Chapter 6).

4.2.3 Day convolution

Just as sum and Cartesian product were seen to arise from the same construction applied to different monoids, both partitional and arithmetic product arise from *Day convolution*, applied to different monoidal structures on $\mathbf{B}$.

It is worth first briefly mentioning the definition of an *enriched category*, which is needed here and also in §4.3. Enriched categories are a generalization of categories where the *set* of morphisms between two objects is replaced by an *object* of some other category.

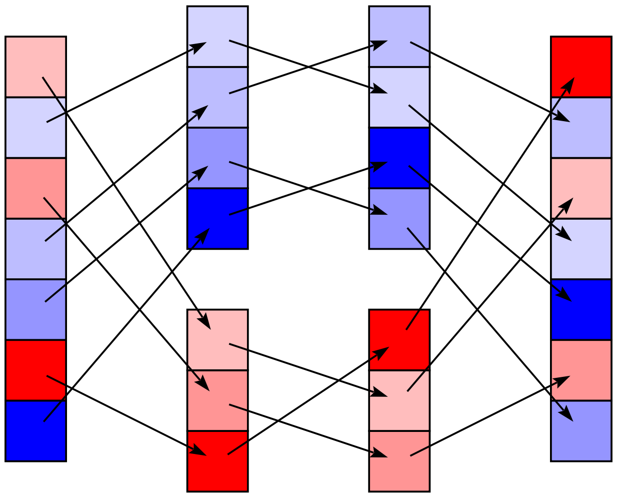


Figure 4.11: $\mathsf{Fin}\ (m+n) \xrightarrow{\sim} \mathsf{Fin}\ m \uplus \mathsf{Fin}\ n \xrightarrow{\sim} \mathsf{Fin}\ m \uplus \mathsf{Fin}\ n \xrightarrow{\sim} \mathsf{Fin}\ (m+n)$

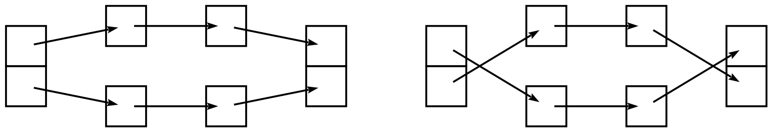


Figure 4.12: Distinct choices of φ that result in identical permutations f

$(\mathsf{Fin}\ m)! \to (\mathsf{Fin}\ n)! \to (\mathsf{Fin}\ (m+n))!$, is neither injective nor surjective. It is not injective since, for example, with $m = n = 1$, there are two distinct inhabitants of $\mathsf{Fin}\ 2 \xrightarrow{\sim} \mathsf{Fin}\ 1 + \mathsf{Fin}\ 1$, but both give rise to the same function $(\mathsf{Fin}\ 1)! \to (\mathsf{Fin}\ 1)! \to (\mathsf{Fin}\ 2)!$ (Figure 4.12), namely, the one which constantly returns the identity permutation (which, indeed, is the only such function which is functorial).

Neither is Ω surjective. Consider the case where $m = n = 2$, and the function $f : (\mathsf{Fin}\ 2)! \to (\mathsf{Fin}\ 2)! \to (\mathsf{Fin}\ 4)!$ given by the table:

	id	(12)
id	id	$(12)(34)$
(12)	(12)	(34)

It is not hard to verify that f is functorial; for example, $f\ id\ (12)\ ;\ f\ (12)\ id = (12)(34)\ ;\ (12) = (34) = f\ (12)\ (12)$. However, we will show that f cannot be of the form $f\ \sigma\ \tau = \varphi \circ (\sigma \uplus \tau) \circ \varphi^{-1}$ for any φ.

For a permutation ψ, denote by $\mathrm{Fix}(\psi) = \{x \mid \psi(x) = x\}$ the set of fixed points of ψ, and by $\mathrm{fix}(\psi) = \#\,\mathrm{Fix}(\psi)$ the number of fixed points. Note that $\mathrm{fix}(\sigma \uplus \tau) = \mathrm{fix}(\sigma) + \mathrm{fix}(\tau)$, since if some value $\mathsf{inl}\ s$ is fixed by $\sigma \uplus \tau$, then s must be fixed by σ, and conversely (and similarly for inr and τ). We also note the following lemma:

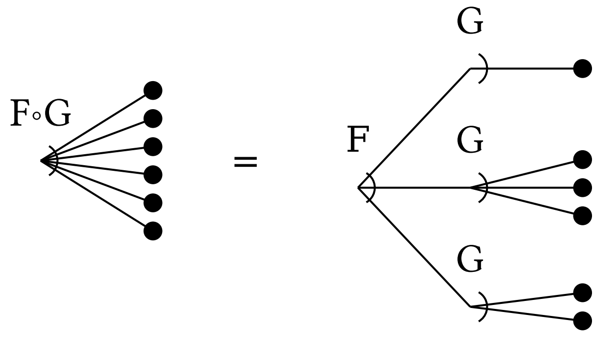


Figure 4.13: Generic species composition

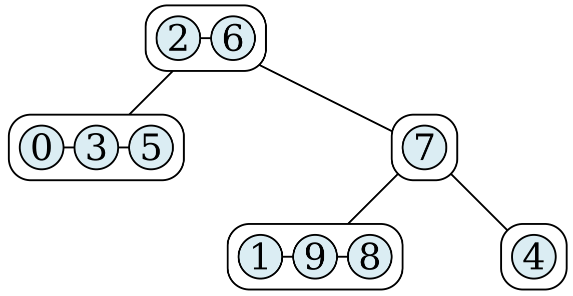


Figure 4.14: An example $(\mathsf{B} \circ \mathsf{L}_+)$-shape

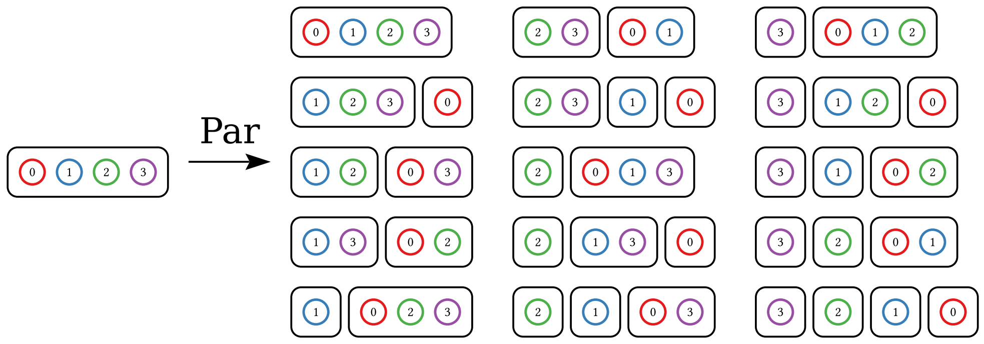


Figure 4.15: The species Par of partitions

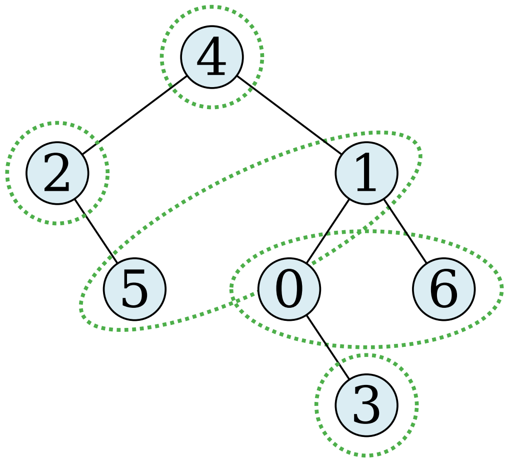


Figure 4.16: An example $(\mathsf{B} \times \mathsf{Par})$-shape

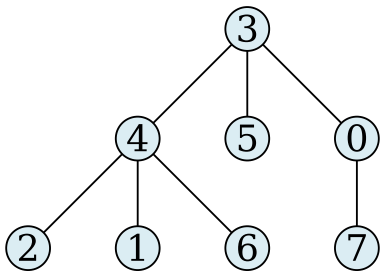


Figure 4.17: An example R-shape

That is, a set partition is a set of *non-empty* sets. Similarly, the species S of permutations is given by $\mathsf{S} = \mathsf{E} \circ \mathsf{C}$, a set of *cycles*.

Given the species Par, we may define the species $\mathsf{B} \times \mathsf{Par}$ of *partitioned trees*. Structures of this species are labeled binary tree shapes with a superimposed partitioning of the labels (as illustrated in Figure 4.16), and can be used to model trees containing data elements with decidable equality; the partition indicates equivalence classes of elements.

Example. The species R of nonempty n-ary ("rose") trees, with data stored at internal nodes, may be defined by the recursive species equation

$$\mathsf{R} = \mathsf{X} \cdot (\mathsf{L} \circ \mathsf{R}).$$

An example R-shape is shown in Figure 4.17. Note the use of L means the children of each node are linearly ordered. Using E in place of L yields a more graph-theoretic notion of a rooted tree, with no structure imposed on the neighbors of a particular node.

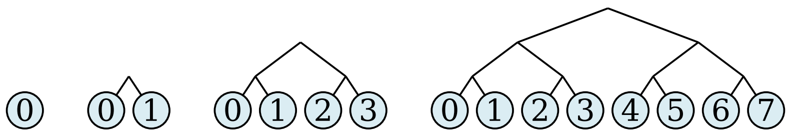


Figure 4.18: Example P-shapes

$\mathsf{Plan} = \mathsf{X} \cdot (\mathsf{C} \circ \mathsf{R})$ is the species of *planar embeddings* of rooted trees, where the top-level subtrees of the root are ordered cyclically. Each node other than the root, on the other hand, still has a linear order on its children, fixed by the distinguished edge from the node towards the root.

Example. The species P of *perfect trees*, with data stored in the leaves, may be defined by

$$\mathsf{P} = \mathsf{X} + (\mathsf{P} \circ \mathsf{X}^2).$$

That is, a perfect tree is either a single node, or a perfect tree containing *pairs*. Functional programmers will recognize this as a *non-regular* or *nested* recursive type; it corresponds to the Haskell type:

$$\textbf{data}\ \mathsf{P}\ a = \mathsf{Leaf}\ a \mid \mathsf{Branch}\ (\mathsf{P}\ (a, a))$$

Figure 4.18 illustrates some example P-shapes.

In addition to being the identity for $\boxtimes$, X is the (two-sided) identity for $\circ$ as well. We have

$$(\mathsf{X} \circ G)\ L = \sum_{\pi \vdash L} \mathsf{X}\ \pi \times \prod_{p \in \pi} G\ p,$$

in which $\mathsf{X}\ \pi$ is $\varnothing$ (cancelling the summands in which it occurs) except in the case where π is the singleton partition $\{L\}$, in which case the summand is isomorphic to $G\ L$. On the other side,

$$(F \circ \mathsf{X})\ L = \sum_{\pi \vdash L} F\ \pi \times \prod_{p \in \pi} \mathsf{X}\ p;$$

the only way to get a product in which none of the $\mathsf{X}\ p$ are $\varnothing$ is when $\pi \cong L$ is the complete partition of L into singleton subsets, in which case we again have something isomorphic to $F\ L$.

As for generating functions, the mapping from species to egfs is indeed homomorphic with respect to composition:

$$(F \circ G)(x) = F(G(x)).$$

A direct combinatorial proof can be given, making use of *Faà di Bruno's formula* [Johnson, 2002].

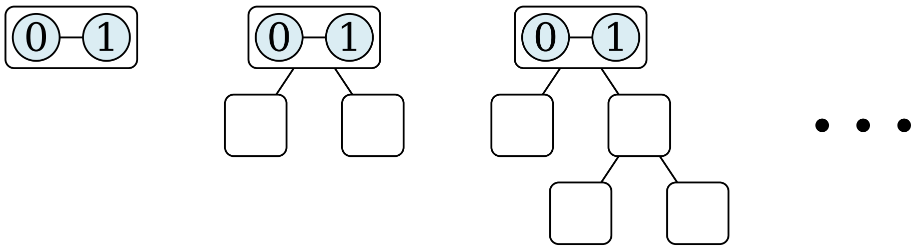


Figure 4.19: An infinite family of $(\mathsf{B} \circ \mathsf{L})$-shapes of size 2

On the other hand, in general,

$$\widetilde{(F \circ G)}(x) \neq \widetilde{F}(\widetilde{G}(x)).$$

Bergeron et al. [1998, Exercise 1.4.3] pose the specific counterexample of $\widetilde{\mathsf{S}}(x) \neq \widetilde{\mathsf{E}}(\widetilde{\mathsf{C}}(x))$, which is not hard to show (hint: $\widetilde{\mathsf{S}}(x) = \prod_{k \geqslant 1} \frac{1}{1-x^k}$ and $\widetilde{\mathsf{C}}(x) = x + x^2 + x^3 + \cdots = \frac{x}{1-x}$). A more intuitive explanation of the failure of ogfs to be homomorphic with respect to composition—along with a characterization of the situations when homomorphism does hold—is left to future work. In any case, to compute ogfs for composed species, one may turn to *cycle index series*, which can be seen as a generalization of both egfs and ogfs, and which retain more information than either; see Bergeron et al. [1998, §1.2, §1.4] for details.

As hinted previously, the formal definition of composition given in (4.3.1) requires additional qualification; in particular, it requires delicate treatment with regard to partitions and infinite families of shapes. To see the issue, let B and L be the species of binary trees and lists. Consider the species $\mathsf{B} \circ \mathsf{L}$, whose shapes should consist of binary trees containing lists at their nodes. Intuitively, this gives rise to infinite families of shapes such as those illustrated in Figure 4.19, which are all of size 2.

There are several possible reactions to Figure 4.19, depending on the exact setting in which we are working.

- If we are working in $(\mathbf{B} \Rightarrow \mathbf{Set})$, where the set of shapes on a given set of labels may be infinite, then this should be allowed, and is exactly the meaning that composition ought to have in this case. Note, however, that this means we would need to allow $\pi \vdash L$ to include "partitions" with arbitrary numbers of *empty* parts (to correspond to the empty lists). Typically, the notation $\pi \vdash L$ denotes partitions into *nonempty* parts.

- On the other hand, if we are working in $(\mathbf{B} \Rightarrow \mathbf{FinSet})$, as is more traditional in a combinatorial setting, this *must not* be allowed. One possibility would be to simply insist that $\pi \vdash L$ in (4.3.1) excludes partitions with empty parts, as is usual. But as we have just seen, this does not generalize nicely to $(\mathbf{B} \Rightarrow \mathbf{Set})$, and in any case it would still be a bit strange, since, for example, $\mathsf{B} \circ \mathsf{L}$ and

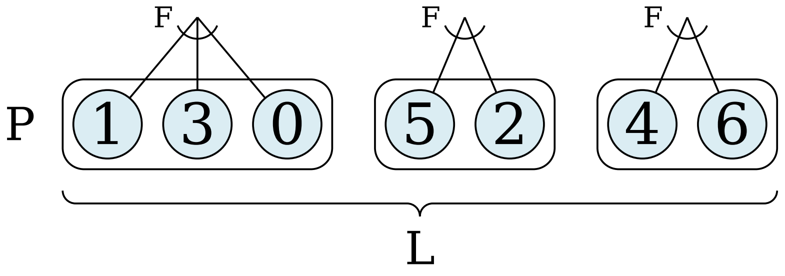


Figure 4.20: Indexed species product

$\beta : B \xrightarrow{\sim} B'$, and $\gamma : C \xrightarrow{\sim} C'$, then $\Pi_{|\mathbf{3}|}(\alpha, \beta, \gamma)$ is the isomorphism $\alpha \times \beta \times \gamma : A \times B \times C \xrightarrow{\sim} A' \times B' \times C'$.

We can now define a general notion of indexed species product. For a species $F : \mathbf{B} \Rightarrow \mathbf{Set}$ and $K \in \mathbf{B}$ a finite set, $F^K : \mathbf{B} \Rightarrow \mathbf{Set}$ represents the $\#K$-fold partitional product of F, indexed by the elements of K (see Figure 4.20):

$$F^K\ L = \exists (P : \mathbf{B}^K).\ \mathbf{B}(\Sigma P, L) \times \Pi(F \circ P).$$

Note that K is regarded as a discrete category, so $P \in \mathbf{B}^K$ is a K-indexed collection of finite sets. $\mathbf{B}(\Sigma P, L)$, a bijection between the coproduct of P and L, witnesses the fact that P represents a partition of L; the coend means there is only one shape per fundamentally distinct partition. The composite $F \circ P = K \xrightarrow{P} \mathbf{B} \xrightarrow{F} \mathbf{Set}$ is a K-indexed collection of F-structures, one on each finite set of labels in P; the Π constructs their product.

It is important to note that this is functorial in K: the action on a morphism $\sigma : K \xrightarrow{\sim} K'$ is to appropriately compose σ with P.

The composition $F \circ G$ can now be defined by

$$(F \circ G)\ L = \exists K.\ F\ K \times G^K\ L.$$

This is identical to the definition given in (4.3.3), except that $G^{\#K}$ has been replaced by G^K, which explicitly records a mapping from elements of K to G-shapes.

This explicit construction relies on a number of specific properties of $\mathbf{B}$ and $\mathbf{Set}$, and it is unclear how it should generalize to other functor categories. Fortunately, in the particular case of $\mathcal{B} \Rightarrow \mathcal{S}$, in HoTT, this more complex construction is not necessary. The anafunctor $G^- : \mathbf{B} \to \mathbf{Spe}$ discussed earlier corresponds in HoTT to a regular functor $G^- : \mathcal{B} \to (\mathcal{B} \Rightarrow \mathcal{S})$: in a symmetric monoidal category, the $(\#K)$-ary tensor product of G is unique up to isomorphism, which in an h-category corresponds to actual equality.

More generally, if we focus on the high-level definition

$$(F \circ G)\ L = \exists K.\ F\ K \times G^K\ L,$$

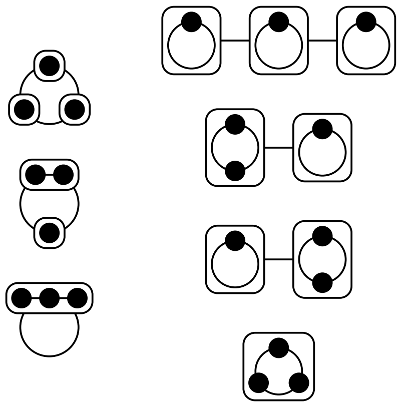


Figure 4.21: $(\mathsf{C} \circ \mathsf{L})$- and $(\mathsf{L} \circ \mathsf{C})$-forms of size 3

leaving the definition of G^K abstract, we can enumerate the properties required of a general functor category $\mathfrak{L} \Rightarrow \mathfrak{S}$ to accommodate it: for starters, $\mathfrak{S}$ must have coends over $\mathfrak{L}$, and $(\mathfrak{S}, \times)$ must be monoidal. We can also say that, whatever the definition of G^K, it will involve partitional product—so we must add in all the requirements for that operation, enumerated in §4.2.3. In fact, this already covers the requirements of $\mathfrak{S}$ having coends and a monoid $\times$, so any functor category $\mathfrak{L} \Rightarrow \mathfrak{S}$ which supports partitional product already supports composition as well.

For a formal proof that composition is associative, see Kelly [2005, pp. 5–6], although some reflection on the intuitive idea of composition should be enough to convince informally: for example, a tree which contains cycles-of-lists is the same thing as a tree-of-cycles containing lists.

Unlike the other monoidal structures on **Spe** (sum and Cartesian, arithmetic, and partitional product), composition is not symmetric. For example, as illustrated in Figure 4.21, there are different numbers of $(\mathsf{C} \circ \mathsf{L})$-forms and $(\mathsf{L} \circ \mathsf{C})$-forms of size 3, and hence $\mathsf{C} \circ \mathsf{L} \not\cong \mathsf{L} \circ \mathsf{C}$.

Proposition 4.3.1. $(\mathbf{Spe}, \circ, \mathsf{X})$ *is monoidal.*

Proof. We have already seen that $\circ$ is associative and that X is an identity for composition. For formal proofs in a more generalized setting see, again, Kelly [2005]. ∎

Like associativity, the right-distributivity laws

$$(F + G) \circ H \cong (F \circ H) + (G \circ H)$$
$$(F \cdot G) \circ H \cong (F \circ H) \cdot (G \circ H)$$

are easy to grasp on an intuitive level. Their formal proofs are not too difficult; the second specifically requires an isomorphism $G^{K_1+K_2} \cong G^{K_1} \cdot G^{K_2}$, which ought to hold no matter what the definition of G^K. The reader may also enjoy discovering why the corresponding left-distributivity laws are false (although they do correspond to species *morphisms* rather than isomorphisms).

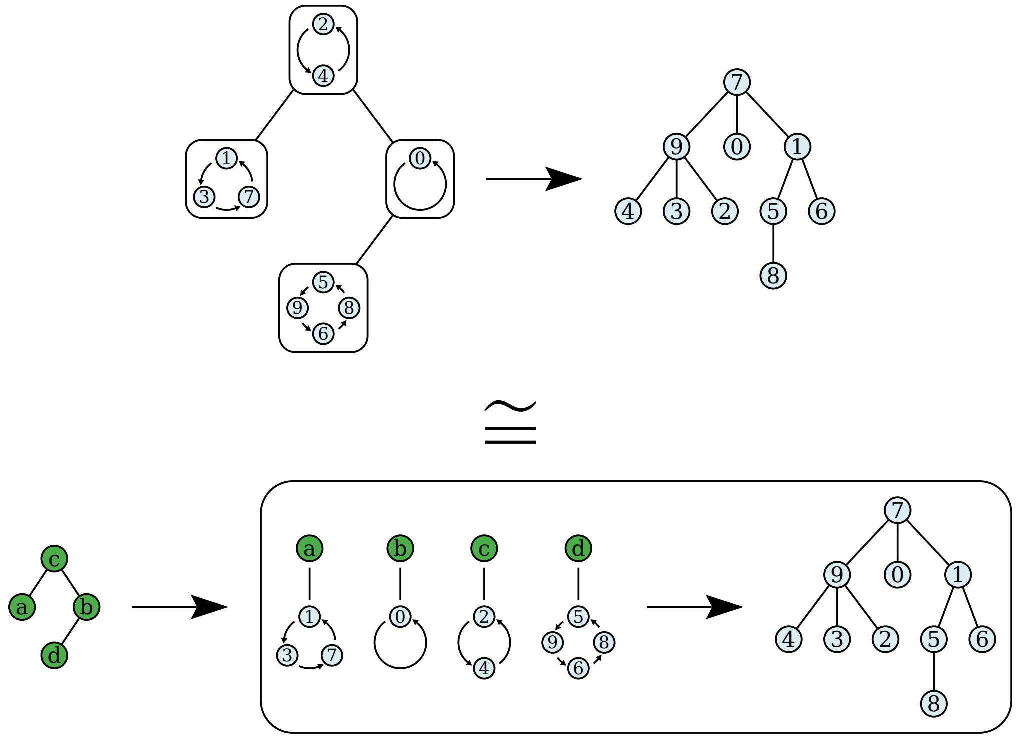


Figure 4.22: Internal Hom for composition

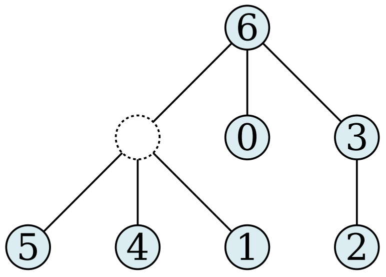


Figure 4.23: An example B'-shape

(§4.5.4). Finally, this notion of higher derivatives paves the way for discussing the internal Hom functors for partitional and arithmetic product (§4.5.5).

There is much more that can be said about differentiation [Menni, 2008, Labelle and Lamathe, 2009, Piponi, 2010b,a, Stay, 2014, McBride, 2012]; in general, there seems to remain a great deal of rich material on differentiation waiting to be explored.

4.5.1 Differentiation in $\mathbf{B} \Rightarrow \mathbf{Set}$

Formally, we create a "hole" by adjoining a new distinguished label to the existing set of labels:

Definition 4.5.1. The *derivative* F' of a species F is defined by

$$F'\ L = F\ (L \uplus \{\star\}).$$

The transport of relabellings along the derivative is defined as expected, leaving the distinguished label alone and transporting the others.

In other words, an F'-shape on the set of labels L is an F-shape on L plus one additional distinguished label. It is therefore slightly misleading to draw the distinguished extra label as an indistinct "hole", as in Figure 4.23, since, for example, taking the derivative twice results in two *different, distinguishable* holes. But thinking of "holes" is still a good intuition for most purposes.

Example. Denote by $\mathfrak{a}$ the species of *unrooted* trees, *i.e.* trees in the pure graph-theoretic sense of a collection of vertices and unoriented edges with no cycles. Also let $\mathcal{A} = \mathsf{X} \cdot (\mathsf{E} \circ \mathcal{A})$ denote the species of rooted trees (where each node can have any number of children, which are unordered). It is difficult to get a direct algebraic handle on $\mathfrak{a}$; however, we have the relation

$$\mathfrak{a}' \cong \mathsf{E} \circ \mathcal{A},$$

since an unrooted tree with a hole in it is equivalent to the set of all the subtrees connected to the hole (Figure 4.24). Note that the subtrees connected to the hole

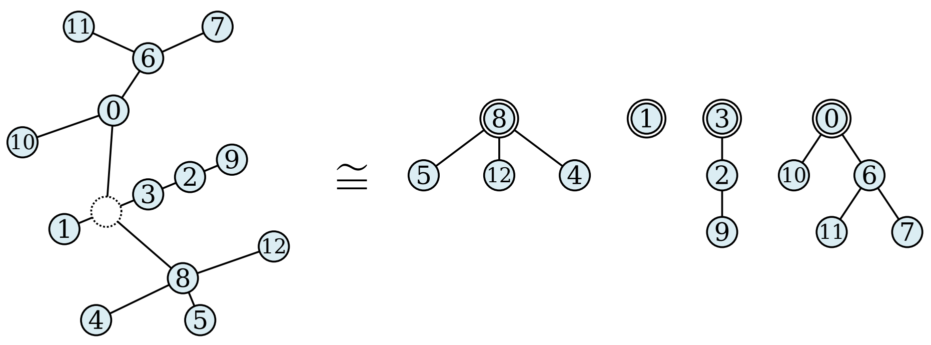


Figure 4.24: $\mathfrak{a}' \cong \mathsf{E} \circ \mathcal{A}$

become *rooted* trees; their root is distinguished by virtue of being the node adjacent to the hole.

Example. $\mathsf{C}' \cong \mathsf{L}$, as illustrated in Figure 4.25.

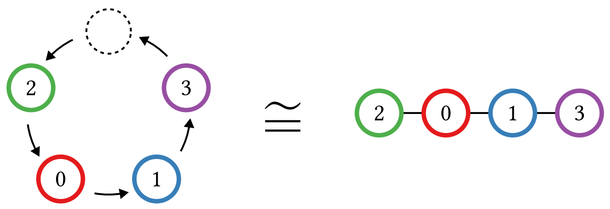


Figure 4.25: $\mathsf{C}' \cong \mathsf{L}$

Example. $\mathsf{L}' \cong \mathsf{L}^2$, as illustrated in Figure 4.26.

Example. Well-scoped terms of the (untyped) lambda calculus may be represented as shapes of the species

$$\Lambda = \varepsilon + \Lambda^2 + \Lambda'.$$

Recall that $\varepsilon = \mathsf{X} \cdot \mathsf{E}$ is the species of elements. (This example appears implicitly—without an explicit connection to species—in the work of Altenkirch et al. [2010], and earlier also in that of Altenkirch and Reus [1999] and Fiore et al. [2003].) Labels

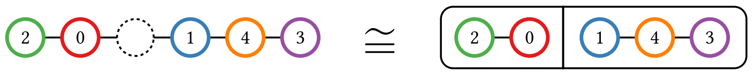


Figure 4.26: $\mathsf{L}' \cong \mathsf{L}^2$

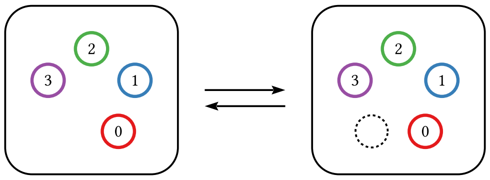


Figure 4.27: The trivial up and down operators on E

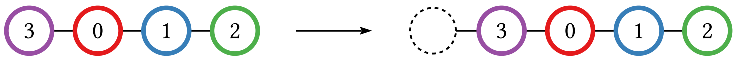


Figure 4.28: An up operator on L

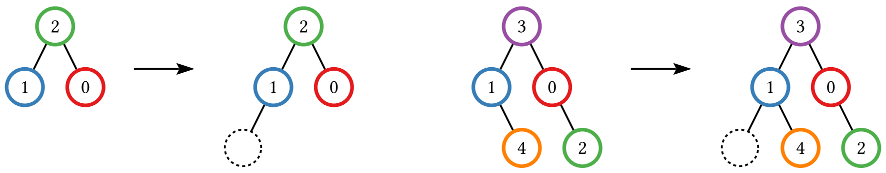


Figure 4.29: An up operator on B

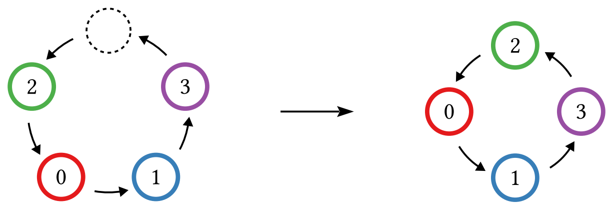


Figure 4.30: A down operator on C

Example. We can similarly make an up operator for the species B of binary trees, which adds a hole as the leftmost (or rightmost) leaf (Figure 4.29).

Example. The species C of cycles, on the other hand, has no up operator. Recall that $\mathsf{C}' = \mathsf{L}$; there is no way to define a *natural* map $\varphi : \mathsf{C} \to \mathsf{L}$. As a counterexample, consider

$$\begin{tikzcd}
\mathsf{C}\ \mathbf{2} \arrow[r, "\varphi_{\mathbf{2}}"] \arrow[d, "\mathsf{C}\ \sigma"'] & \mathsf{L}\ \mathbf{2} \arrow[d, "\mathsf{L}\ \sigma"] \\
\mathsf{C}\ \mathbf{2} \arrow[r, "\varphi_{\mathbf{2}}"'] & \mathsf{L}\ \mathbf{2}
\end{tikzcd}$$

where $\mathbf{2} = \{0, 1\}$ is a two-element set, and $\sigma : \mathbf{2} \xrightarrow{\sim} \mathbf{2}$ is the permutation that swaps 0 and 1. The problem is that $C\ \sigma$ is the identity on $\mathsf{C}\ \mathbf{2}$, but $\mathsf{L}\ \sigma$ is not the identity on $\mathsf{L}\ \mathbf{2}$, so this square cannot possibly commute.

Generalizing from this example, one intuitively expects that there is no up operator whenever taking the derivative breaks some symmetry, as in the case of C. Formalizing this intuitive observation is left to future work.

Down operators are defined dually, as one would expect:

Definition 4.5.3. A *down operator* on a species F is a species morphism $d : F' \to F$.

Example. Again, E has a trivial down operator, which is the inverse of its up operator.

Example. Although we saw previously that the species C of cycles has no up operator, it has an immediately apparent down operator, namely, the natural map $\mathsf{C}' \to \mathsf{C}$ which removes the hole from a cycle, that is, which glues together the two ends of a list.

Example. The species L of linear orders also has an apparent down operator, which simply removes the hole.

Remark. Aguiar and Mahajan [2010, p. 275, Example 8.56] define a down operator on L which removes the hole *if* it is in the leftmost position, and "sends the order to 0" otherwise. However, this seems bogus. First of all, it is not clear what is meant by 0 in this context; assuming it denotes the empty list, it is not well-typed, since species morphisms must be label-preserving.

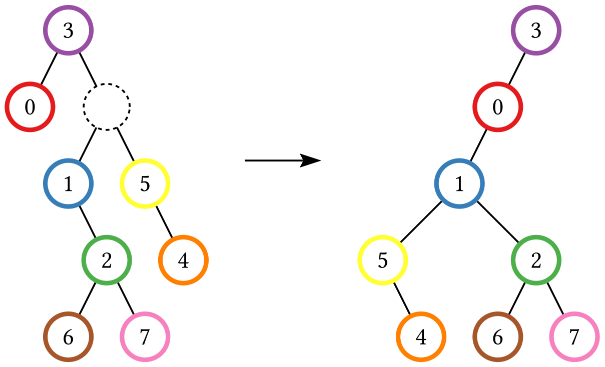


Figure 4.31: An example down operator on B, via stacking

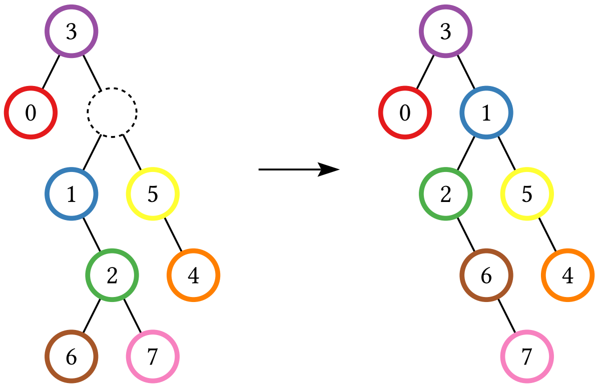


Figure 4.32: An example down operator on B, via promotion

Example. It takes a bit more imagination, but it is not too hard to come up with examples of down operators for the species B of binary trees. For example, the two subtrees beneath the hole can be "stacked", with the first subtree added as the left-most leaf of the remaining tree, and the other subtree added as *its* leftmost leaf (Figure 4.31), or nodes could be iteratively promoted to fill the hole, say, preferring the left-hand node when one is available (Figure 4.32).

These operators are somewhat reminiscent of deletion from data structures such as binary search trees or heaps. Those algorithms rely on a linear order on the labels, and hence do not qualify as natural species morphisms. However, they do indeed qualify as down operators on the **L**-species of binary search trees and heaps, respectively (see §2.4.3 and §5.5).

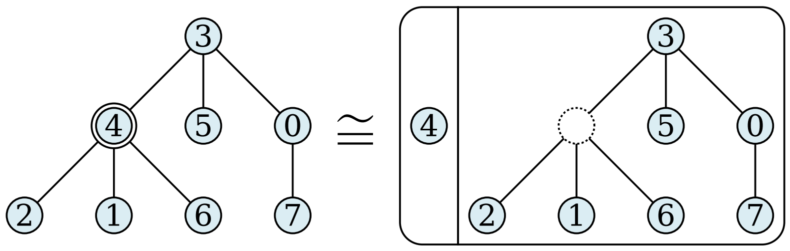


Figure 4.33: Species pointing

4.5.3 Pointing

Definition 4.5.4. The operation of *pointing* can be defined in terms of the species of elements, $\varepsilon = \mathsf{X} \cdot \mathsf{E}$, and Cartesian product:

$$F^{\bullet} = \varepsilon \times F.$$

As illustrated on the left-hand side of Figure 4.33, an $F^{\bullet}$-structure can be thought of as an F-structure with one particular distinguished element.

As is also illustrated in Figure 4.33, pointing can also be expressed in terms of differentiation,

$$F^{\bullet} \cong \mathsf{X} \cdot F'.$$

Similar laws hold for pointing as for differentiation; they are left for the reader to discover.

4.5.4 Higher derivatives

Aguiar and Mahajan [2010, §8.11] describe a generalization of species derivatives to "higher derivatives". The idea of higher derivatives in the context of functions of a single variable should be familiar: the usual derivative is the *first* derivative, and by iterating this operation, one obtains notions of the second, third, ... derivatives. More abstractly, we generalize from a single notion of "derivative", f', to a whole family of higher derivatives $f^{(n)}$, parameterized by a *natural number* n.

Note that taking the derivative of a polynomial reduces the degree of all its terms by one. More generally, the nth derivative reduces the degrees by n. According to the correspondence between species and generating functions, the *degrees* of terms in a generating function correspond to the *sizes* of label sets. Recall that the general principle of the passage from generating functions to species is to replace natural number sizes by finite sets of labels having those sizes. Accordingly, just as higher derivatives of generating functions are parameterized by a natural number which acts on the degree, higher derivatives of species are parameterized by a finite set which acts on the labels.

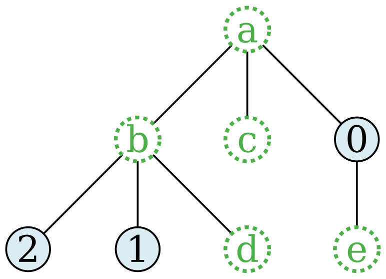


Figure 4.34: An example $\mathsf{B}^{(K)}$-shape

Definition 4.5.5. For a species F and finite set K, the K-derivative of F is defined by

$$F^{(K)}\ L = F\ (K \uplus L).$$

As should be clear from the above discussion, the exponential generating function corresponding to the K-derivative of F is

$$(F^{(K)})(x) = F^{(\#K)}(x),$$

i.e. the $(\#K)$-th derivative of F. Note that we recover the simple derivative of F by setting $K = \{\star\}$. Note also that $F^{(\varnothing)} = F$.

An $F^{(K)}$-shape with labels L is an F-shape populated by both L *and* K. The occurrences of labels from K can be thought of as "K-indexed holes", since they do not contribute to the size. For example, an "$F^{(K)}$-shape of size 3" consists of an F-shape with three labels that "count" towards the size, as well as one "hole" for each element of K. Figure 4.34 illustrates a $\mathsf{B}^{(K)}$-shape of size 3, where $K = \{a, b, c, d, e\}$.

Higher derivatives generalize easily to any functor category $\mathfrak{L} \Rightarrow \mathfrak{S}$ where $(\mathfrak{L}, \oplus, I)$ is monoidal; we simply define

$$F^{(K)}\ L := F\ (K \oplus L).$$

4.5.5 Internal Hom for partitional and arithmetic product

As promised, we now return to consider the existence of an internal Hom functor corresponding to partitional product. We are looking for some

$$- \Rightarrow_{\bullet} - : \mathbf{Spe}^{\text{op}} \times \mathbf{Spe} \to \mathbf{Spe}$$

for which

$$(F \cdot G \Rightarrow_{\mathbf{Spe}} H) \cong (F \Rightarrow_{\mathbf{Spe}} (G \Rightarrow_{\bullet} H)). \tag{4.5.1}$$

Intuitively, this is just like currying—although there are labels to contend with which make things more interesting.

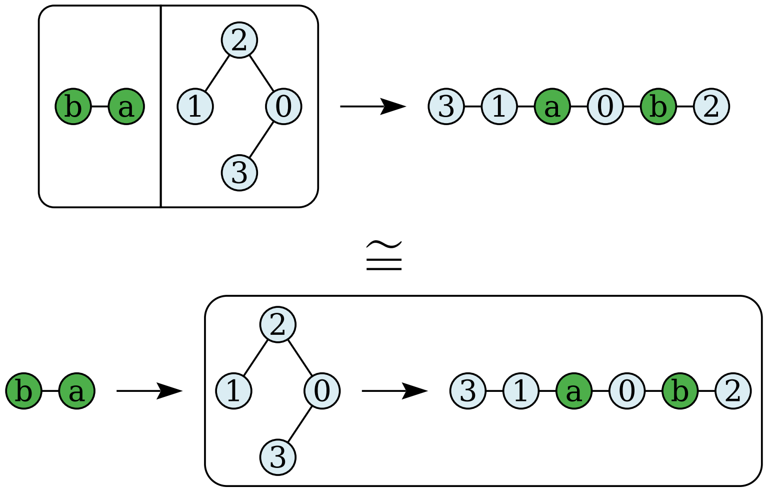


Figure 4.35: "Currying" for partitional product of species

Recall that an $(F \cdot G)$-shape on L is a partition $L_1 \uplus L_2 = L$ together with shapes from $F\ L_1$ and $G\ L_2$. Another way of saying this is that an $(F \cdot G)$-shape consists of an F-shape and a G-shape on two different sets of labels, whose disjoint union constitutes the label set for the entire product shape. Thus, a morphism out of $F \cdot G$ should be a morphism out of F, which produces another morphism that expects a G and produces an H on the disjoint union of the label sets from the F- and G-shapes.

This can be formalized using the notion of higher derivatives developed in the previous subsection. In particular, define $- \Rightarrow_{\bullet} -$ by

$$(G \Rightarrow_{\bullet} H)\ L := G \Rightarrow_{\mathbf{Spe}} H^{(L)}.$$

That is, a $(G \Rightarrow_{\bullet} H)$-shape with labels taken from L is a species morphism, *i.e.* a natural, label-preserving map, from G to the L-derivative of H. This definition is worth rereading a few times since it mixes levels in an initially surprising way—the *shapes* of the species $G \Rightarrow_{\bullet} H$ are *species morphisms* between other species. However, this should not ultimately be too surprising to anyone used to the idea of higher-order functions; it corresponds to the idea that functions can output other functions.

Thus, a $(G \Rightarrow_{\bullet} H)$-shape with labels from L is a natural function that takes a G-shape as input and produces an H-shape which uses the disjoint union of L and the labels from G. This is precisely what is needed to effectively curry a species morphism out of a product while properly keeping track of the labels, as illustrated in Figure 4.35. The top row of the diagram illustrates a particular instance of a species morphism from $\mathsf{L} \cdot \mathsf{B}$ to L. The bottom row shows the "curried" form, with a species morphism that sends a list to another species morphism, which in turn sends a tree to a higher derivative of a list, containing holes corresponding to the original list.

Formally, we have the adjunction (4.5.1). The same result appears in Kelly [2005]

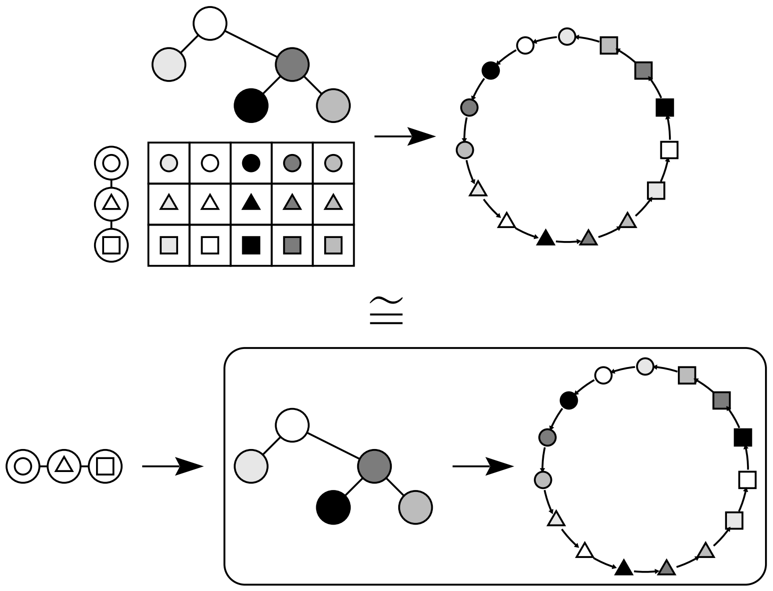


Figure 4.36: "Currying" for arithmetic product of species

4.6 Regular, molecular and atomic species

We now consider the three related notions of *regular*, *molecular*, and *atomic* species. *Regular* species, roughly speaking, are those that correspond to algebraic data types in Haskell or OCaml. A first characterization is as follows:

Definition 4.6.1. The class of *regular* species consists of the smallest class containing $\mathsf{0}$, $\mathsf{1}$, and X, and closed under (countable) sums and products.

There are a few apparent differences between regular species and algebraic data types. First, programming languages do not actually allow *infinite* sums or products. For example, the species

$$\mathsf{X}^2 + \mathsf{X}^3 + \mathsf{X}^5 + \mathsf{X}^7 + \mathsf{X}^{11} + \dots$$

of prime-length lists is a well-defined regular species, but is not expressible as, say, a data type in Haskell[4]. Second, Haskell and OCaml also allow recursive algebraic data types. However, this is not a real difference: the class of regular species is also closed under least fixed points (any implicit recursive definition of a species can in theory be unfolded into an infinite sum or product). Essentially, recursion in algebraic data types can be seen as a tool that allows *some* infinite sums and products to be encoded via finite expressions.

However, there is a more abstract characterization of regular species which does a better job of capturing their essence. We first define the *symmetries* of a structure.

[4]At least not in Haskell 2010.

Recall that $\mathcal{S}_n$ denotes the *symmetric group* of permutations on n elements under composition.

Definition 4.6.2. A permutation $\sigma \in \mathcal{S}_n$ is a *symmetry* of an F-shape $f \in F\ L$ if and only if σ fixes f, that is, $F\ \sigma\ f = f$.

Example. The C-shape in the upper left of Figure 4.37 has the cyclic permutation (01234) as a symmetry, because applying it to the labels results in the same cycle (in the upper right). On the other hand, (12) is not a symmetry; it results in the cycle on the lower left, which is not the same as the original cycle.

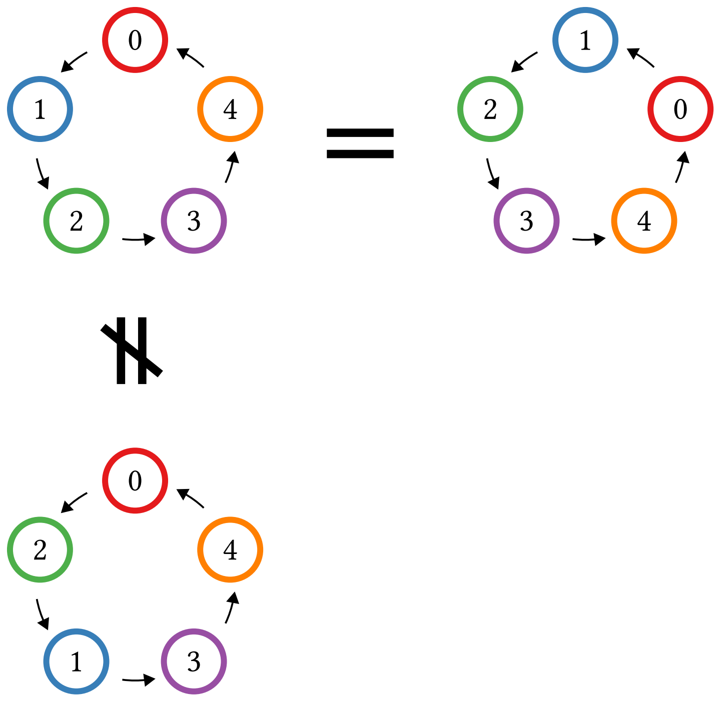


Figure 4.37: A symmetry and a non-symmetry of a C-shape

Example. An L-shape has no nontrivial symmetries: applying any permutation other than the identity to a linear order results in a different linear order.

Example. An E-shape has all possible symmetries: applying any permutation to the labels in a set results in the same set.

We can now state the more abstract definition of regular species; this definition and Definition 4.6.1 turn out to be equivalent.

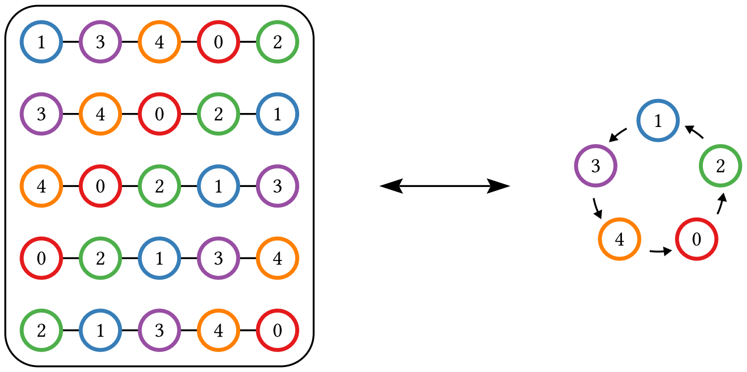


Figure 4.38: Isomorphism between $\mathsf{L}_5/\mathbb{Z}_5$ and C_5

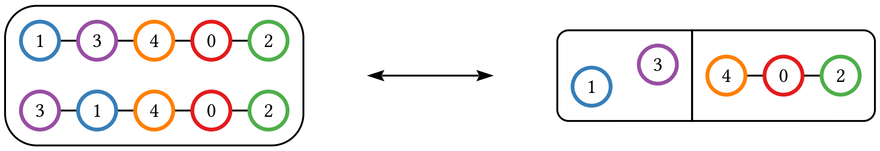


Figure 4.39: Isomorphism between $\mathsf{L}_{\geqslant 2}/\mathbb{Z}_2$ and $E_2 \cdot \mathsf{L}$

In particular, this means that, up to isomorphism, molecular species of size n are in one-to-one correspondence with conjugacy classes of subgroups of $\mathcal{S}_n$. This gives a complete classification of molecular species. For example, it is not hard to verify that there are four conjugacy classes of subgroups of $\mathcal{S}_3$, yielding the four molecular species illustrated in Figure 4.40. The leftmost is X^3, corresponding to the trivial group. The second is $\mathsf{X} \cdot \mathsf{E}_2$, corresponding to the subgroups of $\mathcal{S}_3$ containing only a single swap. The third is C_3, corresponding to $\mathbb{Z}_3$. The last is E_3, corresponding to $\mathcal{S}_3$ itself.

This can in fact be extended to a classification of all species: up to isomorphism, every species has a unique decomposition as a sum of molecular species. As a very simple example, the molecular decomposition of L is

$$\mathsf{L} = 1 + \mathsf{X} + \mathsf{X}^2 + \mathsf{X}^3 + \dots.$$

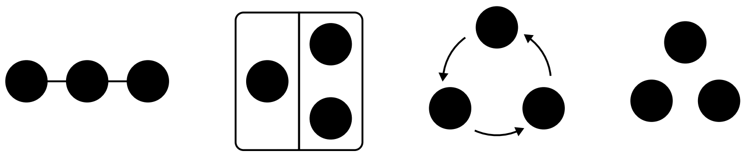


Figure 4.40: The four molecular species of size 3

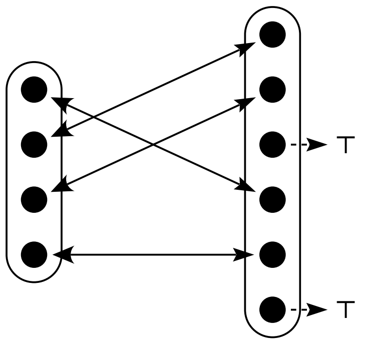


Figure 5.1: A typical copartial bijection

Proposition 5.2.2. *The round-trip laws given in Definition 5.2.1 are equivalent to*

$$\forall ab.\ (f^{\rightarrow}\ a = b) \leftrightarrow (\mathsf{inr}\ a = f^{\leftarrow}\ b). \tag{5.2.1}$$

Proof. Since the laws in question are all mere propositions, it suffices to show that they are logically equivalent; moreover, since the right-to-left direction of (5.2.1) is precisely the second round-trip law, it suffices to show that the left-to-right direction is logically equivalent to the first round-trip law. In one direction, (5.2.1) implies the first round-trip law, by setting $b = f^{\rightarrow}\ a$. Conversely, given the first round trip law,

$$\begin{array}{ll} \quad f^{\rightarrow}\ a = b & \\ \rightarrow & \{\ \text{apply } f^{\leftarrow} \text{ to both sides}\ \} \\ \quad f^{\leftarrow}\ (f^{\rightarrow}\ a) = f^{\leftarrow}\ b & \\ \leftrightarrow & \{\ \text{first round-trip law}\ \} \\ \quad \mathsf{inr}\ a = f^{\leftarrow}\ b. & \end{array}$$

SDG

As an aid in discussing copartial bijections we define $\mathsf{pInv}(f)$ which together with $f : A \to B$ constitutes a copartial bijection $A \subseteq B$.

Definition 5.2.3. A *partial inverse* $\mathsf{pInv}(f)$ to $f : A \to B$ is defined so that

$$(A \subseteq B) \equiv (f : A \to B) \times \mathsf{pInv}(f),$$

that is,

$$\mathsf{pInv}(f) :\equiv (g : B \to \top + A) \times (\forall ab.\ (f\ a = b) \leftrightarrow (\mathsf{inr}\ a = g\ b)).$$

We also define some notation to make working with copartial bijections more convenient.

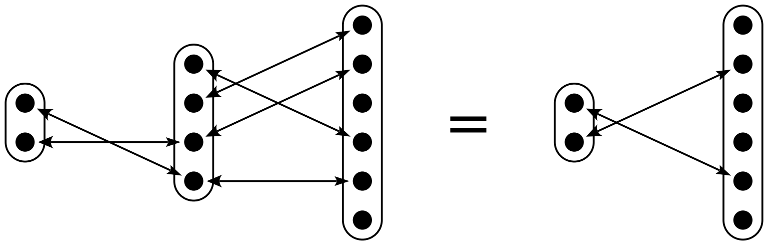


Figure 5.2: Composition of copartial bijections

Definition 5.2.4. First, for any types A and B, there is a canonical copartial bijection $A \subseteq A + B$, which we denote simply by inl; similarly, $\mathsf{inr} : B \subseteq A + B$.

For the remainder, assume there is a copartial bijection $p : K \subseteq L$.

- $p^{\rightarrow}\ K$ denotes the image of K under p, that is, the set of values in L in the range of $p^{\rightarrow}$; we often simply write $p\ K$ instead of $p^{\rightarrow}\ K$.
- $p|_K : K \xrightarrow{\sim} p\ K$ denotes the bijection between K and the image of K in L.
- When some $q : K' \subseteq K$ is understood from the context, we also write $p|_{K'}$ as an abbreviation for $(p \circ q)|_{K'}$, the bijection between K' and the image of K' in L under the composite $(p \circ q)$.
- $p_\top = \{l : L \mid l^{\leftarrow} = \mathsf{inl}\ \star\}$ denotes the "extra" values in L which are not in the image of K.
- $\tilde{p}$ denotes the canonical bijection $K + p_\top \xrightarrow{\sim} L$.

We now turn to the category structure on copartial bijections.

Proposition 5.2.5. *Copartial bijections compose, that is, there is an associative operation*

$$- \circ - : (B \subseteq C) \to (A \subseteq B) \to (A \subseteq C).$$

Proof. This can be intuitively grasped by studying a diagram such as the one shown in Figure 5.2.

More formally, we set $(g \circ f)^{\rightarrow} = g^{\rightarrow} \circ f^{\rightarrow}$ and $(g \circ f)^{\leftarrow} = f^{\leftarrow} \bullet g^{\leftarrow}$, where $- \bullet -$ denotes Kleisli composition for the $\top + -$ monad (*i.e.* $(<=<) :: (b \to \mathsf{Maybe}\ c) \to (a \to \mathsf{Maybe}\ b) \to (a \to \mathsf{Maybe}\ c)$ in Haskell). Associativity thus follows from the associativity of function composition and Kleisli composition. In the following proof we also make use of $(-)^* : (A \to \top + B) \to (\top + A \to \top + B)$, *i.e.* $(=\!\!<\!\!<)$ in Haskell.

To show the required round-trip property we reason as follows.

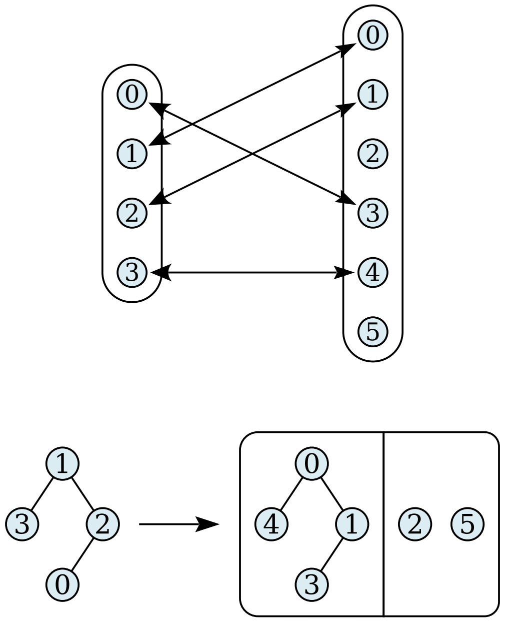


Figure 5.3: Lifting a strictly copartial bijection

isomorphisms in $\mathcal{B}_\subseteq$, and functors necessarily preserve isomorphisms, so bijections on labels are still sent to bijections between structures.

The case of strictly copartial bijections, that is, $K \subseteq L$ where $\#K < \#L$, is more interesting. Each structure in the set $F\ K$, with labels in K, must map to a structure in $F\ L$, given an embedding of K into L. Intuitively, this can be thought of as introducing extra labels which must be incorporated into the structure in a suitably canonical way. However, the copartial bijection $p : K \subseteq L$ affords no structure whatsoever on the extra labels (that is, those $l \in L$ for which $p^{\leftarrow}\ l = \mathsf{inl}\ \star$). So it is not acceptable, for example, to prepend the extra labels to the front of a list structure, since there is no canonical way to choose an ordering on the extra labels. The only feasible approach is to simply attach the extra labels in a *set*, as illustrated in Figure 5.3.

Moreover, note that one cannot adjoin a *new* set of labels with every lift. Performing multiple lifts would then result in multiple sets of extra labels (*e.g.* a list of such sets), but this fails to be functorial, since separately lifting two copartial bijections (resulting in a list of two extra sets) would be different than lifting their composition (resulting in only one). So the only option is to have *every* copartial species structure accompanied by a set of "extra" labels (which may be empty). Transporting along a strictly copartial bijection results in some labels being added to the set.

Intuitively, every normal species F gives rise to a copartial species $F_\subseteq$ which acts like the species $F \cdot \mathsf{E}$. In fact, along these lines we can formally define a fully faithful embedding of $\mathbf{Spe}$ into $\mathbf{Spe}_\subseteq$.

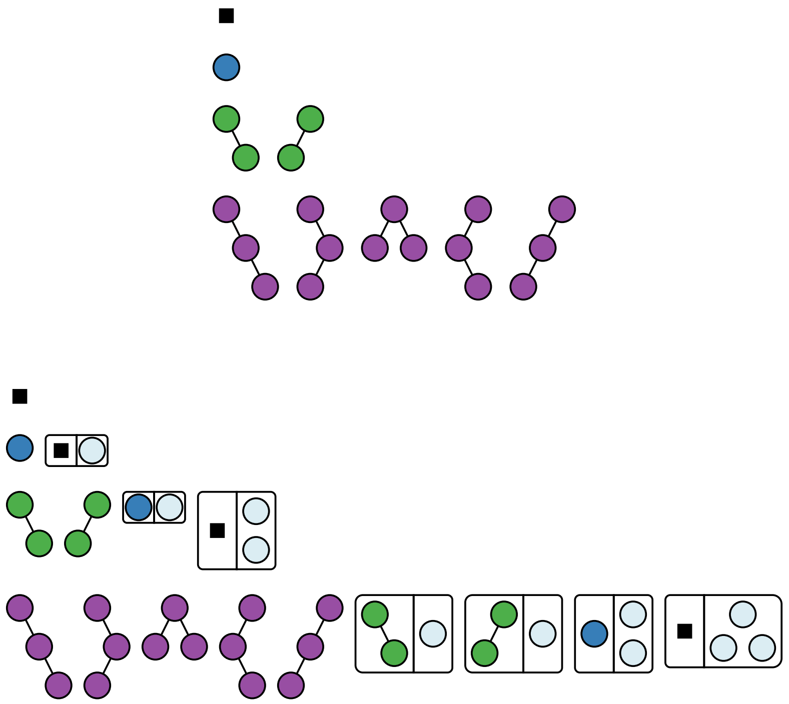


Figure 5.4: $\mathsf{B} \cdot \mathsf{E}$ (bottom) is the prefix sum of B (top)

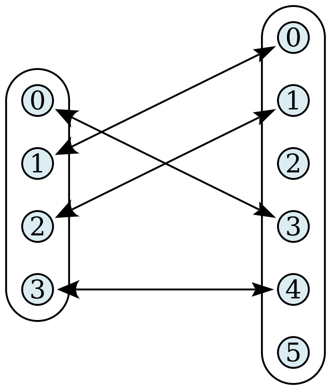
Note that as an operator on generating functions, this has an inverse, given by

$$(b_0 + b_1 x + b_2 x^2 + \dots) \mapsto (b_0 + (b_1 - b_0)x + (b_2 - b_1)x^2 + \dots). \tag{5.2.2}$$

Consider, then, an attempted proof that $-_{\subseteq} : \mathbf{Spe} \to \mathbf{Spe}_{\subseteq}$ is essentially surjective. Given a copartial species $S \in \mathbf{Spe}_{\subseteq}$, this would require us to produce some $F \in \mathbf{Spe}$ such that $F_{\subseteq} \cong S$. If we think of $F_{\subseteq}$ as intuitively acting like $F \cdot \mathsf{E}$, we see that S should correspond to a "prefix sum" of F. Then we should ideally be able to construct F via an operation similar to (5.2.2). That is (passing to $\mathbf{P} \Rightarrow \mathbf{Set}$ and $\mathcal{P}_{\hookrightarrow} \Rightarrow \mathcal{S}$), we would define

$$\begin{aligned} F\ 0 &:= S\ 0 \\ F\ (n+1) &:= S\ (n+1) - S\ n. \end{aligned}$$

However, we must make sense of this subtraction. We cannot simply take a set difference (indeed, set difference makes no sense in the context of HoTT). What is needed is some sort of canonical injection $\iota : S\ n \hookrightarrow S\ (n+1)$, in which case we could make sense of $S\ (n+1) - S\ n$ as the elements of $S\ (n+1)$ not in the image of ι. In the case of species of the form $F \cdot \mathsf{E}$, there indeed exists such a canonical injection, which sends each shape in $(F \cdot \mathsf{E})\ n$ to the same shape with the extra label n adjoined to the set. The whole point, however, is that we are *trying to prove* that every $S \in \mathbf{Spe}_{\subseteq}$ is of



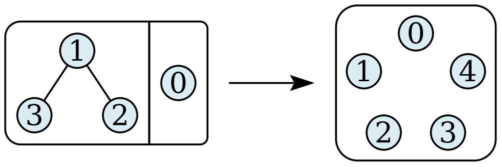


Figure 5.5: A copartial species which loses information

this form. We must therefore come up with some injection $S\ n \hookrightarrow S\ (n+1)$ without any intensional knowledge about S.

This is precisely where we get stuck. There is, of course, a canonical injection $\mathsf{Fin}\ n \hookrightarrow \mathsf{Fin}\ n+1$, but the functoriality of S only guarantees that this lifts to a *function* $S\ n \to S\ (n+1)$—functors may preserve isomorphisms, but in general, they need not preserve monomorphisms. This insight guides us to a counterexample. Consider the $(\mathcal{B}_{\subseteq} \Rightarrow \mathcal{S})$-species whose shapes *of size* 5 *or smaller* consist of a binary tree paired with a set, and *on larger sizes* simply consist of a set (Figure 5.5).

One may verify that this does, in fact, describe a valid functor $\mathcal{B}_{\subseteq} \to \mathcal{S}$. However, it does not preserve information: above size 5 the shapes all collapse, and information about smaller shapes is lost. The intuition that all copartial species shapes must come equipped with a set of labels is correct, in a sense, but there is some latitude in the way the rest of the shape is handled.

We may also, therefore, consider the subcategory $\mathcal{B}_{\subseteq} \hookrightarrow \mathcal{S}$ of *monomorphism-preserving* functors $\mathcal{B}_{\subseteq} \Rightarrow \mathcal{S}$; along the lines sketched above, we can indeed prove an equivalence between this category and **Spe**. At present, the pros and cons of considering $\mathbf{Spe}_{\subseteq}$ versus this subcategory are not clear to me.

Finally, we consider which of the properties from Table 5.1 hold for $\mathcal{B}_{\subseteq} \Rightarrow \mathcal{S}$.

- Since the target category is just $\mathcal{S}$ we automatically get all the properties required of $\mathfrak{S}$ alone (*e.g.* monoidal, complete, and Cartesian closed); $\mathcal{S}$ is also cocomplete and so has coends over $\mathcal{B}_{\subseteq}$.

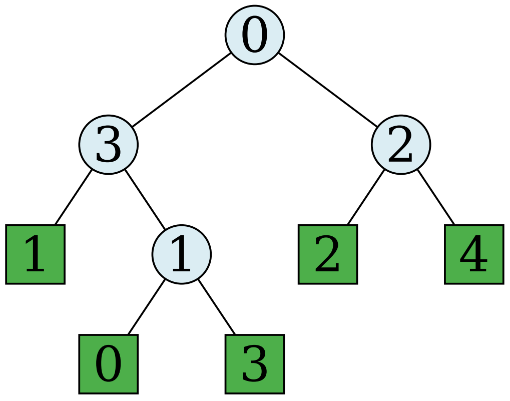


Figure 5.6: A two-sort species of binary trees

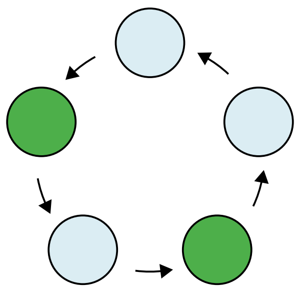


Figure 5.7: A bicolored cycle

the two-sort species of binary trees with internal nodes and leaves labelled by distinct sorts. Figure 5.6 illustrates an example shape of this species, with one label sort represented by blue circles, and the other by green squares.

Example. $\mathsf{C} \circ (\mathsf{X} + \mathsf{Y})$ is the species of *bicolored cycles*, *i.e.* cycles whose labels are colored with one of two colors (Figure 5.7).

Example. Recall the example from §4.5.1, in which open terms of the untyped lambda calculus are modeled using the species

$$\Lambda = \varepsilon + \Lambda^2 + \Lambda'.$$

However, this species does not correctly model the size of lambda calculus terms. §4.5.1 suggested instead using the multisort species

$$\Xi = \mathsf{X} \cdot \mathsf{E} + \mathsf{Y} \cdot \Xi^2 + \mathsf{Y} \cdot \frac{\partial}{\partial \mathsf{X}} \Xi.$$

Chapter 6

Labelled structures

Now that we have a foundation for describing labelled shapes, the next step is to extend them into full-blown *data structures* by adjoining mappings from labels to data. For example, Figure 6.1 illustrates an example of a labelled shape together with a mapping from the labels to data values. However, this must be done in a principled way which allows deriving properties of labelled structures from corresponding properties of labelled shapes. This chapter begins with an overview of *Kan extensions* (§6.1) and *analytic functors* (§6.2), which provide the theoretical basis for extending labelled shapes to labelled structures. It then continues to briefly discuss introduction and elimination forms for labelled structures, and outline some directions for future work.

6.1 Kan extensions

The definition of analytic functors, given in §6.2, makes central use of the notion of a (left) *Kan extension*. This section defines Kan extensions and provides some useful intuition for understanding them in this context. For more details, see Mac Lane

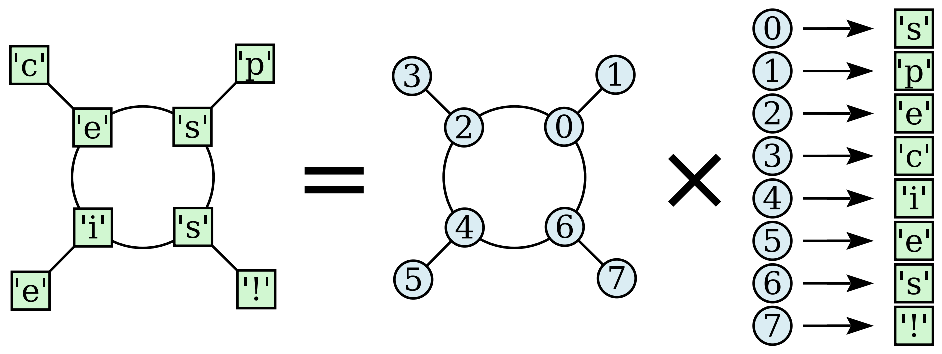


Figure 6.1: Data structure = shape + data