Symmetric Containers

by

Håkon Robbestad Gylterud

Thesis for the degree of

Master of Science

(Master i Matematikk)

Department of Mathematics
Faculty of Mathematics and Natural Sciences
University of Oslo

November 2011

Contents

Introduction

I was first introduced to the idea of a differential operator on data structures through its practical application in Haskell programming. A data structure can be imagined to have a set of *shapes*, and each shape has a set of *positions* where you can insert data. If you differentiate a data structure you get the data structure with all possible ways to remove a position from a shape. This is used to create data structures, called Zippers, where you can traverse downwards in the structure, while maintaining enough data to traverse back up again.

The Wikibook on Haskell (`http://en.wikibooks.org/wiki/Haskell/`), which was my primary source of information on this subject back then, has a lovely analogy with Ariadne's thread. In their story Theseus, by Ariadne's advice, instead of using a thread, uses a Zipper structure to backtrack his way through the labyrinth. I instantly fell for the beauty of the theory, where you could use familiar techniques from calculus to analyse and write functional programs.

When I started working with this thesis, I had a vague idea of what anti-derivatives of data structures could be. Differentiating them means to select a position and remove it. In analogy anti-derivatives should somehow lack information about where positions were located, so that in order to remove an element you would have to give information about how the positions were located.

This idea of lack of information lead me at first to look at intuitionistic set theory, especially the categorical variety topos theory. I hoped that since intuitionistic logic is the logic of what an ideal mathematician could "know", undecidable statements could be used to model the lack of information I felt should be present in anti-derivatives. But soon I started employing more combinatorial techniques, laying aside the intuitionistic logic. In retrospect I can see that the machinery I ended up using, namely groupoids, has a close relationship to intuitionistic logic. The category of groupoids is in fact a model of intentional Martin-Löf type theory (Martin Hofmann and Streicher 1998).

My thesis is based upon the work in (Michael Abbott, Thorsten Altenkirch, Neil Ghani and Conor Mcbride 2008), which gives a mathematical foundation to the study of differentiating data structures. Their central notion is that of a *container*. For the ease of reading this thesis, the fundamentals container theory is placed in the beginning of the first chapter. The first chapter also contains the development of Taylor series of containers. Formal power series are powerful combinatorial tools and the development of Taylor series of containers was mentioned as possible future work in (Michael Abbott et al. 2008).

At the same time I that was reading papers about containers, I was also reading about the process of categorification and higher categories. This inspired me to categorify the notion of a container, which proved to be an interesting excursion to the field of 2-categories. The second chapter of this thesis, with its definition of a

categorical container, reflects this work. I prove that the 2-category of categorical containers has a full and faithful embedding into the 2-category of endofunctors on the 2-category of small categories.

In the third chapter I keep the 2-category perspective, but specialise to a more moderately categorified notion of containers, namely symmetric containers. I prove that finite, decidable containers have anti-derivatives in the category of symmetric containers.

The mathematical machinery used in this thesis is a mix of category theory, combinatorics, logic and group theory. All proofs in this thesis are concrete constructions of some sort. This is because the matter at hand is of a constructive nature, and because I feel that a proof by construction is more tangible than a proof by logical argument. Although the proofs are constructive, they are not written in a formal type system.

One thing the reader should be aware when reading this thesis is that the term "group" will consistently refer to a groupoid with a single object, which we denote $*$. Another is that the underlying set-theory of the category *Set* of sets and functions is presumed to be intuitionistic, something like the free topos specified in (Joachim Lambek and Scott 1986). However, the only consequence of this is that we distinguish between sets which are decidable and those which are not.

I would like to thank my advisor, Dag Normann, for his good advice and patience. I also want to thank Kristian Moi, for all the fruitful discussions we have had about topics which have greatly influenced this thesis, and Sigurd Segtnan for having an open door at his office; in through which I have often peeked and got answer to my questions. In the weeks leading up to my deadline I had good help from people reading my thesis, correcting both mathematical and grammatical, as well as typographical mistakes. For this I would like to thank Eivind, John, Sigurd and my family.

Related work

The notion of a symmetric container is closely related to the notion of a combinatorial species, developed by Joyal and others. This is elaborated in 3.7.

While working with this thesis, I also discovered that John Baez and James Dolan, motivated by quantum physics, have developed the notion of *stuff types*, which is basically symmetric containers on finite sets. This is perhaps most accessible through Simon Byrne's essay (Simon Byrne 2005) on the topic.

Notation

$Ob(\mathcal{C})$ The type of objects in the category C, when no confusion will arise, we may write $A : \mathcal{C}$ (or $A \in \mathcal{C}$ if $\mathcal{C}$ is locally small) instead of $A : Ob(\mathcal{C})$ ($A \in Ob(\mathcal{C})$).

$Mor_{\mathcal{C}}(A, B)$ The type of morphisms (or arrows) from $A : Ob(\mathcal{C})$ to $B : Ob(\mathcal{C})$. When the category in question can be inferred from the context, we may write $f : A \to B$ to say that f is a morphism from A to B, and refer to $Mor_{\mathcal{C}}(A, B)$ as just $A \to B$.

$\circ : Mor_{\mathcal{C}}(B, C) \times Mor_{\mathcal{C}}(A, B) \to Mor_{\mathcal{C}}(A, C)$ The composition of morphisms in the category $\mathcal{C}$. Written in infix notation: $g \circ f$. We suppress any mention of the category $\mathcal{C}$ in this notation as it will be inferable from context.

$id_A : Mor_{\mathcal{C}}(A, A)$ The identity morphism. Often we will write $id_A : A \to A$. We suppress any mention of the category $\mathcal{C}$ in this notation as it will be inferable from context.

Cat The category of small categories. The construction of functor categories makes this into a 2-category.

$Grpd$ The full subcategory of Cat of small groupoids. A groupoid is a category where every arrow $f : A \to B$ has an inverse f^{-1}, s.t. $f \circ f^{-1} = id_B$ and $f^{-1} \circ f = id_A$. We adopt the convention that any group is a groupoid with $1_{Set} = \{*\}$ as its set of objects. And that an action of a groupoid (or group) G (on sets) is a functor $m : G \to Set$.

Set The category of sets. We may regard this as a full subcategory of Cat consisting of discrete categories. Note that sets are small groupoids.

$0_{\mathcal{C}} : Ob(\mathcal{C})$ The initial object in the category $\mathcal{C}$ (when existent). When clear from context the subscript will be dropped.

$\odot_A$ The unique morphism $0 \to A$

$1_{\mathcal{C}} : Ob(\mathcal{C})$ The terminal object in the category $\mathcal{C}$ (when existent). When clear from context the subscript will be dropped.

$\Box_A$ The unique morphism $A \to 1$

$A + B$ The coproduct of $A, B : Ob(\mathcal{C})$, with injections $l_{A,B} : A \to A + B$ and $r_{A,B} : B \to A + B$. Injections are sometimes used as natural transformations.

$f + g$ Co-product of morphisms. Given $f : A \to B$ and $g : C \to D$, then $f + g : A + C \to B + D$.

$A \times B$ The product of $A, B : Ob(\mathcal{C})$, with projections $\pi_{0,A,B} : A \times B \to A$ and $\pi_{1,A,B} : A \times B \to B$. Projections are sometimes used as natural transformations.

$f \times g$ Product of morphisms. Given $f : A \to B$ and $g : C \to D$, then $f \times g : A \times C \to B \times D$.

$\Delta_A : A \to A \times A$ The diagonal morphism. Sometimes used as a natural transformation.

$\nabla_A : A + A \to A$ The co-diagonal morphism. Sometimes used as a natural transformation.

B^A The exponential object of $A, B : Ob(\mathcal{C})$.

$\int_X F(X, X)$ The end of a functor $\mathcal{C} \times \mathcal{C}^{op} \to \mathcal{D}$.

$\sum_{a \in A} B(a)$ The sum over an A-indexed family of sets $B : A \to Set$. Elements of this set is of the form (a, b) where $a \in A$ and $b : B(a)$.

$\prod_{a \in A} B(a)$ The product over an A-indexed family of sets $B : A \to Set$. Elements of this set is of the form $f : A \to \sum_{a:A} B(a)$ such that $f(a) = (a, b)$ for some $b \in B(a)$.

$A \setminus X$ If $X \subseteq A$, then $A \setminus X$ is the set of elements in A which are not in X.

$A - x$ If A is a set, and $x \in A$, then $A - x$ denote the set of elements in A which are not equal to x.

$f|_{x,y}$ If A and B are decidable sets, $x \in A$ and $y \in B$ and $f : A \to B$ is an isomorphism such that $f(x) = y$, then $f|_{x,y} : (A - x) \to (B - y)$ is the isomorphism which results from restricting f.

$A \hookrightarrow B$ The set of injective functions from a set A to a set B. $f : A \hookrightarrow B$ means that f is a function from A to B and is injective.

$\mathbb{N}$ Is the set of natural numbers, including 0. $\mathbb{N}_+ = \mathbb{N} - 0$

$Fin : \mathbb{N} \to Set$ This functor is defined inductively as: $Fin(0) = 0_{Set}$ and $Fin(n + 1) = Fin(n) + 1_{Set}$. Thus $Fin(n)$ is a decidable set of n elements. We may identity $Fin(n)$ with $\{0, 1, \cdots, n - 1\}$.

$\mathrm{Im}(f)$ If $f : A \to B$ is a function, $\mathrm{Im}(f)$ denotes its image.

S_n The n-th symmetric group. We view this as the group isomorphism $Fin(n) \to Fin(n)$.

1 Containers

In this chapter we give an introduction to containers and develop a notion of Taylor series of containers.

1.1 Data structures as functors

While the current thesis is one of mathematics, not of computer science, the concept of a container is motivated as giving a mathematical model of strictly positive data types, which are very much in use when programming. Thus it seems fitting to present this as motivation for the later abstract treatment. The reader content with the study of mathematical abstractions for their own sake may safely skip this subsection.

On an abstract level, one can look at computer programming as being concerned with the manipulation of various data structures. At the bottom you have primitive data types such as boolean values, fixed bit integers and floating point numbers and arrays of bytes. On a higher lever you have tuples, lists, trees, arbitrary size integers, even lists of lists and trees of lists etc. With lazy evaluation you can even manipulate infinite lists and trees to a certain extent.

When programming data structures it is a virtue to make use of generic data structures. What that means is that in stead of having to write separate code for lists of integers and lists floating point numbers, you can just describe lists in general, without regard to what they contain.

Various programming languages let you create generic data structures in various ways. In untyped or dynamically typed languages (such as Lisp or Python), you just have to be careful with what you assume about the content of the data structure. In the statically typed language C++ you have a system of template classes and functions, from which the compiler generates the specialized classes and functions. In Haskell, type variables are a part of a system of algebraic data types, such that generic types are really types in the language.

One way to view generic data structures is as an operator on data structures. The structure of lists is an operator taking for instance the type of integers to the type of lists of integers.

In languages such as Haskell this way of looking at generic data structures is reflected in the syntax:

```
data List x = Empty | Cons x (List x)
```

This is the definition of the list data type in Haskell. The way to read this is "A list with elements among `x` is either empty or it is an element of `x` (the head) and a list (the tail)".

Going from programming to mathematics one often translates data types and instances into sets and elements. Though some set theories, such as ZF, contain huge sets which makes no sense to a computer, the over-all categorical structures are similar enough to pretend that data structures are sets.

(Note: The above claim is only true if we consider programs which terminates, consisting of so called total functions. To deal with non-termination it is more common to translate types into what is called *domains* (for reference see (Thomas Streicher 2006)). But in the current treatment we will gladly restrict our attention to total functions.)

Thus a more mathematical notation for the above, suppressing the names of the constructors (`Empty, Cons`), would be:

$$L(X) \cong 1 + X \times L(X)$$

The 1 represents the singleton set containing only the empty list. $+$ is disjoint union and $\times$ the cartesian product. This is an equation of sets (up to bijections). Note that we are not looking for an X satisfying it, but rather an operation on sets called L, which for every set X satisfy this equation.

Given such an equation there are in particular two solutions which are of special interests, namely the terminal and the initial algebras.

It is important that these operations are uniform. For instance if two sets are in bijective correspondence, we want the corresponding sets of lists to be in bijective correspondence as well. This and much more we get from requiring that these operations are functors.

An endofunctor $F : Set \to Set$ is a rule which assigns every set X another set $F(X)$ and every function $f : X \to Y$ a function $F(f) : F(X) \to F(Y)$, in a way that preserves composition and identity functions.

As an example take $L(X)$ to be the set of all finite sequences with elements in X. So a typical element of $L(x)$ are either the empty list, e, or looks like $(x_0, \cdots, x_{n-1})$ where $n \in \mathbb{N}_+$ and $x_i \in X$ for every $0 \leq i < n$.

Given $f : X \to Y$ we may lift this function to a function on lists in a canonical way. Here defined by induction on the structure of lists:

$$\begin{aligned} &L(f) : L(X) \to L(Y) \\ &L(f)(e) = e \\ &L(f)(x_0 \cdot xs) = f(x_0) \cdot L(f)(xs) \end{aligned}$$

(e being the empty list and $\cdot$ meaning concatenation.)

An advantage of functors is that they are easily composed, so that from the concept of lists and the concept of trees one gets a concept of lists of trees. It can also be extended to several parameters.

In Haskell this idea is an integral part of the way data structures are implemented. They always instantiate the `Functor` type class:

```
class Functor d where
    fmap :: (a -> b) -> (d a -> d b)

instance Functor List where
    fmap f Empty = Empty
    fmap f (Cons h t) = Cons (f h) (fmap f tail)
```

Haskell takes the use of category theory much further than this simple example. For an ingenious use of concept of monads to handle functions with side effects see (Philip Wadler 1990).

1.2 Containers

In this subsection we give a brief introduction to the notion of a container, as defined in (Michael Abbott, Thorsten Altenkirch and Neil Ghani 2005), which will provide the basis for this thesis. We will restrict our attention to containers in one variable ($I = 1_{Set}$ in the notation of (Michael Abbott et al. 2005)).

Definition 1.2.1. A **container** is a pair $(S \rhd P)$ where $S : Set$ and $P : S \to Set$. The elements of the set S are called the **shapes** of the container. Given a shape s, the elements of the set $P(s)$ are called the **positions** of s.

The category of containers, Con, has containers as objects, and morphisms $Mor_{Con}(S \rhd P, T \rhd Q) = \sum_{f:S\to T} \sigma : \prod_{x\in S} Q(f(x)) \to P(x)$.

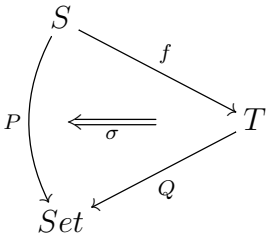


Figure 1.2.1: A diagram of a container morphism (f, σ) between the containers $S \rhd P$ and $T \rhd Q$

A container is like a set of templates where you can fill in information in each position. Given a set X, we may construct the set of ways to fill positions with elements from X. This is given as the set $\sum_{s \in S} X^{P(s)}$, which elements are pairs (s, p) where $s \in S$ is a chosen shape and $p : P(s) \to X$ assigns to each position a value in X.

This construction is functorial:

Definition 1.2.2. Given a container $(S \rhd P)$ we define the functor $[\![S \rhd P]\!] : Set \to Set$, by:

$$[\![S \rhd P]\!](X) = \sum_{s \in S} X^{P(s)}$$
$$[\![S \rhd P]\!](\phi)(s, p) = (s, \phi \circ p)$$

where $\phi : X \to Y$ is a function, thus $[\![S \rhd P]\!](\phi) : [\![S \rhd P]\!](X) \to [\![S \rhd P]\!](Y)$.

Further more a container morphism $(f, \sigma) : (S \rhd P) \to (T \rhd Q)$ gives us a natural transformation $[\![f, \sigma]\!] : [\![S \rhd P]\!] \Rightarrow [\![T \rhd Q]\!]$.

Definition 1.2.3. Given two containers $(S \rhd P)$ and $(T \rhd Q)$, and a container morphism $(f, \sigma) : (S \rhd P) \to (T \rhd Q)$ we define a natural transformation

$$[\![f, \sigma]\!] : [\![S \rhd P]\!] \Rightarrow [\![T \rhd Q]\!]$$
$$[\![f, \sigma]\!]_X (s, p) = (f(s), p \circ \sigma_s))$$

Naturality is easily checked. Given $\phi : X \to Y$ and $(f, \sigma) : (S \rhd P) \to (T \rhd Q)$

$$\begin{aligned}
([\![f, \sigma]\!]_Y \circ [\![S \rhd P]\!](\phi))(s, p) &= [\![f, \sigma]\!]_Y ([\![S \rhd P]\!](\phi)(s, p)) \\
&= [\![f, \sigma]\!]_Y (s, \phi \circ p) \\
&= (f(s), (\phi \circ p) \circ \sigma_s) \\
&= (f(s), \phi \circ (p \circ \sigma_s)) \\
&= [\![T \rhd Q]\!](\phi)(f(s), p \circ \sigma_s) \\
&= [\![T \rhd Q]\!](\phi)([\![f, \sigma]\!]_X (s, p)) \\
&= ([\![T \rhd Q]\!](\phi) \circ [\![f, \sigma]\!]_X)(s, p)
\end{aligned}$$

Thus we conclude that $[\![-]\!] : Con \to Set^{Set}$ is a functor.

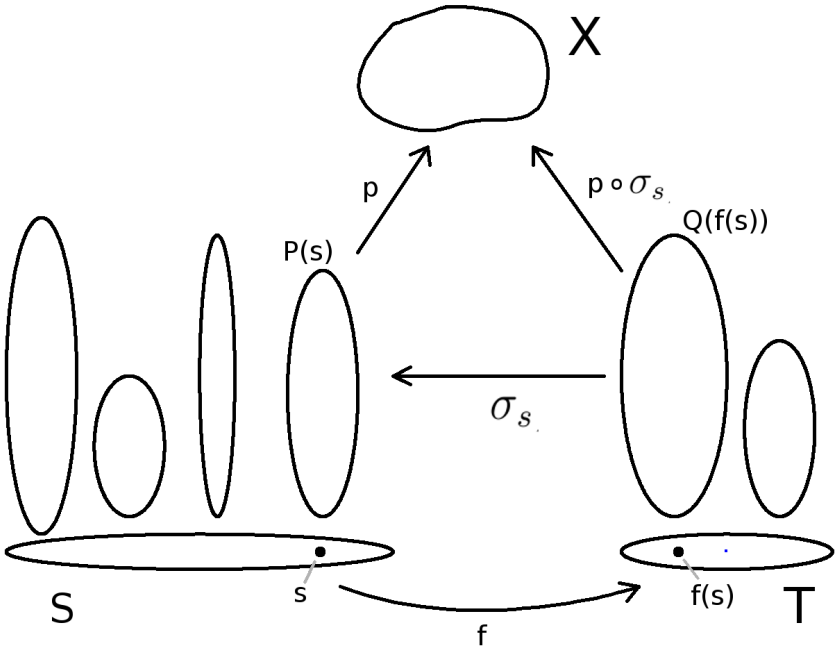


Figure 1.2.2: A figure illustrating how a container morphism (f, σ) lets you translate (s, p) into $(f(s), p \circ \sigma_s)$. Here we picture the sets of positions as stalks above S and T.

1.3 Examples

Here we give some examples of containers with increasing level of sophistication.

Example 1.3.1: The identity container I has a single shape with a single position. The name is suitable since $[\![I]\!] \cong Id_{Set}$.

Example 1.3.2: The pair container I^2 has a single shape $*$ with two positions $\{0, 1\}$. The functor $[\![I^2]\!]$ is naturally isomorphic to the functor $X \mapsto X^2$.

Example 1.3.3: Given a set A, the constant container $K(A)$ has A as its set of shapes, and an empty set of positions in each shape. The functor $[\![K(A)]\!]$ is naturally isomorphic to the constant functor $X \mapsto A$.

Example 1.3.4: The list container $L = (\mathbb{N} \triangleright Fin(-))$. The functor $[\![L]\!]$ maps a set to (a set isomorphic to) the set of finite sequences in X.

Example 1.3.5: The stream container $S = (1 \triangleright P_S)$ where $P_S(*) = \mathbb{N}$. The functor $[\![L]\!]$ is naturally isomorphic to the functor $X \mapsto X^{\mathbb{N}}$

Example 1.3.6: While the set of natural numbers is defined as the initial algebra of the functor $F : Set \to Set$ given by $F(X) = X + 1$, the set of co-natural numbers $\mathbb{N}_\omega$ is the terminal co-algebra of F.

The co-natural numbers are $0, 1, 2, \cdots \omega$. But they do not necessarily form a decidable set. ω is formed by an infinite number of applications of the successor function. Thus it cannot be decided in any finite stage if an element of $\mathbb{N}_\omega$ is ω.

Addition is definable on $\mathbb{N}_\omega$ by co-recursion. We can order the co-natural numbers by saying that

$$x < y \Leftrightarrow \neg\exists z \in \mathbb{N}_\omega \; y + z = x \tag{1.3.1}$$

Further more, the following mapping defines a container with $\mathbb{N}_\omega$ as its shapes:

$$\begin{aligned} P_{L^\infty}\mathbb{N}_\omega &\to Set \\ x &\mapsto \{y \in \mathbb{N}_\omega \mid y < x\} \end{aligned}$$

The container $L^\infty = (\mathbb{N}_\omega \rhd P_{L^\infty})$, is the container of possibly infinite lists. The functor it represents sends a type X to the terminal coalgebra of the functor $F(Y) = 1 + X \times Y$, and is in that respect dual to the list functor.

1.4 Properties of containers

In this section we briefly review what is known about the category of containers.

Products and co-products

We begin by stating the comforting propositions, proved in (Michael Abbott et al. 2005), that the category of containers have all finite products and coproducts.

Proposition 1.4.1. *Given two containers $F = S \rhd P$ and $G = T \rhd Q$, their coproduct is $F + G = (S + T \rhd \nabla_{Set} \circ (P + Q))$. Coproducts are preserved by $[\![-]\!]$.*

Proposition 1.4.2. *Given two containers $F = S \rhd P$and $G = T \rhd Q$, their product is $F \times G = (S \times T \rhd R)$ where*

$$R(s, t) = P(s) + Q(t)$$

Products are preserved by $[\![-]\!]$.

Note the similarity with multiplying monomials, i.e. $sX^p \times tX^q = (t \times s)X^{p+q}$.

Representation

The following theorem, proved in (Michael Abbott et al. 2005), show us that containers are good representations of the functors they are mapped to with $[\![-]\!]$.

Theorem 1.4.3. *$[\![-]\!] : Con \to Set^{Set}$ is fully faithful.*

Cartesian closure

In 2010, it was shown (see (Thorsten Altenkirch, Paul Levy and Staton 2010)), using the representation theorem, that the category of containers is in fact cartesian closed.

Theorem 1.4.4. *Con is a cartesian closed category*

1.5 Differentiating containers

Consider the following "coincidence":

Given a set X, the set of ordered triples with elements of X is X^3. If we were to pick and remove one of the elements in the triple, we would be left with a pair of elements of X, but in possibly three different ways, depending on which of the three elements we remove. So the information we are left with can be represented by the elements of $3 \times X^2$.

The "coincidence" is of course that if we differentiate the expression X^3 we get $3X^2$.

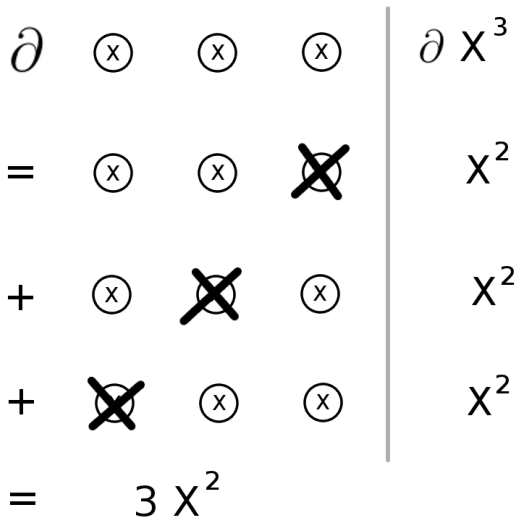


Figure 1.5.3: A figure illustrating the remove an element semantics of differentiation.

If one thinks about it, one can even explain the product rule of differentiation using this remove-an-element semantics. Removing an element from a pair of structures you either remove it from the left one or you remove it from the right one.

$$\partial[F \times G] = \partial[F] \times G + F \times \partial[G] \tag{1.5.1}$$

This correspondence seems also to be valid for recursive expressions. Consider removing an element from a list. It is impossible to do with the empty list, so we can ignore that case. If we remove the first element, we have only the rest of the list. Removing any other element leaves us with the first element and the rest of the list with a hole.

This explains that from the recursive equation

$$L(X) \cong 1 + X \times L(X) \tag{1.5.2}$$

we can differentiate to get the recursive equation

$$\partial L(X) \cong L(X) + X \times \partial L(X) \tag{1.5.3}$$

In (Michael Abbott et al. 2008) Abbott, Altenkirch, Ghani and Mcbride made this interesting correspondence formal using containers. Here we shall review how this is done and look at some at some examples.

The derivative of a container is to be a container composed of all the ways we can remove a position in the container. If $x \in A$ we use the notation $A - x$ to denote the set of elements in A not equal to x. I.e. $A - x = \{a \in A \mid a \neq x\}$.

We do not want to assume that our underlying set theory is classical, but we want to be able to distinguish between positions in our shapes. Otherwise we run into trouble when removing them.

Definition 1.5.1. A container $(S \rhd P)$ is called decidable if for every $s \in S$ we have that $P(s) : Set$ is a decidable set. I.e. $\forall x, y \in P(s)(x = y \vee x \neq y)$ is true internally in Set.

Definition 1.5.2. Let $(S \rhd P) : Con$ be a decidable container. Its **derivative** $\partial(S \rhd P) : Con$ is given by $\partial(S \rhd P) = (S' \rhd P')$ where P' and S' are given by:

$$S' = \sum_{s:S} P(s)$$
$$P'(s, p) = P(s) - p$$

The name differentiation of this operation hints to the fact that it satisfy the rules we normally associate with differentiation of functions. In (Michael Abbott et al. 2008) it was shown that the following isomorphism hold in the category of containers.

Theorem 1.5.3. *For any decidable containers F and G, the following isomorphism holds:*

$$\partial(F+G) \cong \partial F + \partial G$$
$$\partial(F \times G) \cong \partial F \times G + F \times \partial G$$

1.6 From containers to series

In (Michael Abbott et al. 2008) it is hinted at the possibility of using the differential operator on containers to develop Taylor series for containers. In this section we write out the details of this process, which - to the knowledge of the author - has not been done elsewhere. There is however a combinatorial idea closely related to the idea of containers, called *combinatorial species* originally developed by Joyal and others ((Bergeron, Labelle and Leroux 1998) is a good reference on combinatorial species), where Taylor series play a major role. The relationship between these two notions will be further expanded in the third chapter.

Now let us be inspired by the success of calculus and try to differentiate everything in sight and make Taylor series, for the moment without thought to whether it really makes sense.

Example 1.6.1: As we have stated already, the list container represent a functor with the property that:

$$L(X) \cong 1 + X \times L(X) \tag{1.6.1}$$

From this equation we see that $L(0) \cong 1$. So our first coefficient in the Taylor series is also 1. To obtain the next coefficient, we differentiate the above equation:

$$\partial L(X) \cong L(X) + X \times \partial L(X)$$
$$\partial L(0) \cong 1$$

Another 1 for the series. We write out the next three steps in one batch, in order to spot the pattern:

$$\begin{aligned}
\partial^2 L(X) &\cong \partial L(X) + \partial L(X) + X \times \partial^2 L(X) \\
&\cong 2 \times \partial L(X) + X \times \partial^2 L(X) \\
\partial^3 L(X) &\cong 3 \times \partial^2 L(X) + X \times \partial^3 L(X) \\
\partial^4 L(X) &\cong 4 \times \partial^3 L(X) + X \times \partial^4 L(X)
\end{aligned}$$

(Each equation is obtained by differentiating the one above it)

A simple induction argument gives us $\partial^n L(0) = n!$, and we see that our Taylor series is constant 1. In a speculative moment we might wonder if $L(X) = 1 + X + X^2 + X^3 + \cdots$ in some sense; and in a sense it does: The first coefficient represents the empty list. X represents a list with a single element from X and X^2 the list with two elements from X, and so on. The fact that the coefficients are all ones reflects that for each number of positions, there is only one shape in the list container which has that number of positions. This will be more apparent in the next example.

Example 1.6.2: Inspired by the previous example, let us count the number of strict binary trees with up to five leaf by calculating the first six coefficients in the Taylor series of the binary tree functor:

$$\begin{aligned}
B(X) &\cong X + B(X)^2 \\
\partial B(X) &\cong 1 + 2B(X)\partial B(X) \\
\partial^2 B(X) &\cong 2((\partial B(X))^2 + B(X)\partial^2 B(X)) \\
\partial^3 B(X) &\cong 2(3\partial^2 B(X)\partial B(X) + T(X)\partial^3 B(X)) \\
\partial^4 B(X) &\cong 2(4\partial^3 B(X)\partial B(X) + 3(\partial^2 B(X))^2 + B(X)\partial^4 B(X)) \\
\partial^5 B(X) &\cong 2(5\partial^4 B(X)\partial B(X) + 10\partial^3 B(X)\partial^2 B(X) + B(X)\partial^5 B(X))
\end{aligned}$$

Assuming $B(0) = 0$ (i.e. there is no empty tree), we get the following values:

$$\begin{aligned}
B(0) &\cong 0 \\
\partial T(0) &\cong 1 \\
\partial^2 B(0) &\cong 2 \\
\partial^3 B(0) &\cong 12 \\
\partial^4 B(0) &\cong 96 \\
\partial^5 B(0) &\cong 1680
\end{aligned}$$

Which give the Taylor series $B(X) = X + X^2 + 2X^3 + 5X^4 + 14X^5 + \cdots$. So there is one tree with one leaf node, one with two, two with three, five with four and twenty-four strict binary trees with five leaf nodes. In total there are 23 strict binary trees with up to five leaf nodes. This of course is nothing revolutionary, as it is well known that $1, 1, 2, 5, 14$ is the beginning of the catalan numbers, which counts the number of strict binary trees. But it does seem an odd coincidence that these number occur by the mechanical operation of differentiation.

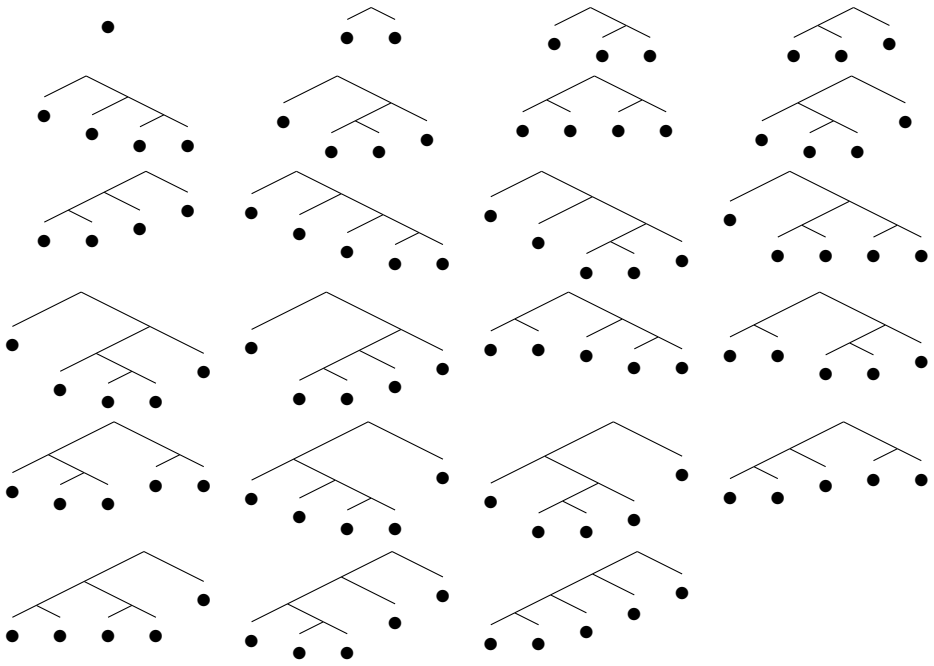


Figure 1.6.4:
A listing of all 23 binary trees with five or less leaf nodes. See the appendix for the code which generated them.

In these two examples we have stated, but in no way proven, that the coefficients of a Taylor series, obtained by differentiating the fix point equations counts the number of shapes with a given number of positions. So let us prove just that!

Taylor series

Our first obstacle is the fact that each coefficient in the Taylor series is a fraction of $n!$. We definitely want every coefficient to be a whole, natural number, so our "division" must be justified in some way.

Lemma 1.6.3. *For all $n \in \mathbb{N}$ and decidable $F = (S \triangleright P) : Con$, we have an isomorphism*

$$\partial^n F \cong (\sum_{s \in S} Fin(n) \hookrightarrow P(s) \triangleright Q)$$

Where Q is given by:

$$Q(s, p) = P(s) \setminus \mathrm{Im}(p)$$

Proof. By induction on n. The case $n = 0$ is trivial. If $\partial^n F \cong (\sum_{s \in S} p : Fin(n) \hookrightarrow P(s) \rhd Q)$, then the set of shapes in $\partial^{n+1} F$ is

$$\sum_{(s,p) \in \sum_{s \in S} Fin(n) \hookrightarrow P(s)} P(s) \setminus \mathrm{Im}(p) \tag{1.6.2}$$

$$\cong \sum_{s \in S} \sum_{p : Fin(n) \hookrightarrow P(s)} P(s) \setminus \mathrm{Im}(p) \tag{1.6.3}$$

$$\cong \sum_{s \in S} Fin(n) + 1 \hookrightarrow P(s) \tag{1.6.4}$$

$$\cong \sum_{s \in S} Fin(n+1) \hookrightarrow P(s) \tag{1.6.5}$$

Let us call the isomorphism from (1.6.2) to (1.6.5) f. Given a shape $((s,p),q)$, we have that the set of positions is

$$P(s) \setminus \mathrm{Im}(p) - q \cong P(s) \setminus \mathrm{Im}(p')$$

where p' is given by $(s,p') = f((s,p),q)$. So we have a natural isomorphism between the sets of positions.

□

Lemma 1.6.4. *For all $n \in \mathbb{N}$ and decidable $F = (S \rhd P) : Con$, we have an isomorphisms*

$$[\![\partial^n F]\!](0) \cong \sum_{s \in S} p : Fin(n) \xrightarrow{\cong} P(s)$$

Proof. Follows from Lemma 1.6.3 and the fact that $P(s)$ is decidable. □

Lemma 1.6.5. *For all $n \in \mathbb{N}$, any decidable container $F = (S \rhd P) : Con$ and any set X, there is a free action of S_n on $[\![\partial^n F]\!](X)$.*

Proof. By lemma 1.6.3, we can assume that

$$[\![\partial^n F]\!](X) = \sum_{s : S} \sum_{p : Fin(n+1) \hookrightarrow P(s)} X^{P(s) \setminus \mathrm{Im}(p)}$$

So given $\sigma \in S_n$, we let $\sigma(s,p,f) = (s, p \circ \sigma, f)$, which is well defined since precomposition with σ preserves the image.

□

Definition 1.6.6. Given a decidable container $F : Con$, we define its **Taylor series** to be $T(F) : \mathbb{N} \to Set$ given by:

$$T(F)(n) = \partial^n F(0)/S_n$$

where $\partial^n F(0)/S_n$ is the quotient of $\partial^n F(0)$ by the action of S_n discussed in Lemma 1.6.5.

Lemma 1.6.7. *If $[\![\partial^n F]\!](0)$ is finite and decidable, so is $[\![\partial^n F]\!](0)/S_n$ and*

$$|\,[\![\partial^n F]\!](0)| = n! \cdot |\,[\![\partial^n F]\!](0)/S_n|$$

Proof. Two elements $x, y \in [\![\partial^n F]\!](0)$ are equal in $[\![\partial^n F]\!](0)/S_n$ iff there is a $\sigma \in S_n$ such that the action of σ maps x to y. Since S_n is finite and $[\![\partial^n F]\!](0)$ is by assumption decidable, we need only search through S_n to decide equality between x and y in $[\![\partial^n F]\!](0)/S_n$.

Now since both $[\![\partial^n F]\!](0)$ and $[\![\partial^n F]\!](0)/S_n$ are finite and decidable, and the action of S_n is free, the statement about the cardinalities is basic group theory. □

This justifies that we may call $T(F)$ a Taylor series for F.

Container from a series

Definition 1.6.8. Given a series of sets $A : \mathbb{N} \to Set$, we define **the container with S as its coefficients** as the container $C(A) = (S \rhd P)$ given by:

$$S = \sum_{n:\mathbb{N}} A(n)$$
$$P((n, x)) = Fin(n)$$

Lemma 1.6.9. *Given a series of sets $A : \mathbb{N} \to Set$, we have a natural isomorphism $T(C(A)) \cong A$.*

Proof. By lemma 1.6.4, $T(C(A))(n)$ is isomorphic to the set of equivalence classes of pairs $[((i, a), p)]$ where $a \in A(n)$ and $p : Fin(n) \cong Fin(i)$. The equivalence is such that $((i, a), p) \sim ((j, b), q)$ iff $i = j$, $a = b$ and there exists $\sigma \in S_n$ such that $p \cong b \circ \sigma$. But if $Fin(n) \cong Fin(i)$ and $Fin(n) \cong Fin(j)$ then $i = j = n$, and such a σ will always exist, so $T(C(A))(n) \cong A(n)$.

□

Definition 1.6.10. For any container $F = (S \rhd P)$, define $N(F) : \mathbb{N} \to Set$ by $N(F)(n) = \{s \in S \mid P(s) \cong Fin(n)\}$.

Now we can formulate what we initially wanted to say:

Theorem 1.6.11. *For any decidable container $F = S \triangleright P$, there is a natural isomorphism $N(F) \cong T(F)$.*

Proof. By lemma 1.6.4, $T(F)(n)$ is isomorphic to the set of equivalence classes of pairs $[(s, p)]$, where $p : Fin(n) \cong P(S)$. Since any two isomorphisms $p, q : P(s) \cong Fin(n)$, will be made equivalent by some $\sigma \in S_n$, $T(F)(n) \cong N(F)(n)$.

□

In other words, the n-th Taylor coefficient for a container is a set which has exactly one element for each shape in the container with n positions.

Analytic containers

We proved that $T(C(A)) \cong A$, but what about $C(T(F))$ how does it compare to F? As the following example shows, they are not isomorphic in general.
Example 1.6.12: Let S be the stream container as discussed in Example 1.3.5. No matter how many times we iterate the differential operator on this container, there will be no empty shapes. So $T(S)(n) = 0$ for all n. Thus $C(T(S))$ is the zero container which has no shapes. Since S clearly has a shape, these two are not isomorphic.

This is in clear analogy to the situation in classical real analysis, where not every infinitely differentiable function is equal to its Taylor expansion about 0. It is this analogy which inspires the following definition.

Definition 1.6.13. Let F be a decidable container. We say that F is **analytic** if $F \cong C(T(F))$.

We happen to be so lucky that we can use Theorem 1.6.11 to fully characterise the analytic containers. Remember that the theorem says that the Taylor series of a container collects the shapes with finite positions. It is thus no wonder that if a container is fully given by its Taylor series, it has only finite containers.

Theorem 1.6.14. *A decidable container $F = (S \triangleright P)$ is analytic iff for every shape $s \in S$ the set $P(s)$ is finite (i.e. isomorphic to $Fin(n)$ for some $n \in \mathbb{N}$).*

Proof. Assume that F is analytic and let $(f, \sigma) : F \to C(T(F))$ be an isomorphism. Fix a shape s. By definition of $C(-)$ we have that $f(s) = (n, a)$ for some $n \in \mathbb{N}$ (and $a \in T(F)(n)$). Thus $\sigma_s : Fin(n) \to P(s)$ is the desired isomorphism.

For the other way around, we assume that there are chosen isomorphisms $\phi_s : Fin(n_s) \to P(s)$. We can then construct an isomorphism $(f, \sigma) : F \to C(N(F))$,

$$f(s) = (n_s, s)$$
$$\sigma_s = \phi_s$$

.

The function $f : S \to \sum_{n\in\mathbb{N}}\{s \in S \mid P(s) \cong Fin(n)\}$ is well defined by our assumption that $P(s) \cong Fin(n_s)$. Since both f and all ϕ_s are isomorphisms of sets it is clear that (f, s) is an isomorphism of containers. By Theorem 1.6.11 we are done.

□

1.7 Anti-derivatives?

So far, every concept from analysis which we have tried to import to containers has been a success: Differentiation, Taylor series and analytic functions. Let us consider something which at first appearance seems to not work out well, namely anti-derivation.

In analysis every analytic function has anti-derivatives. To find an anti-derivative, you simply apply the rules of differentiation backwards on each term in the Taylor series of the function. For instance X^2 becomes $\frac{1}{3}X^3$. A crucial observation here is that even though the Taylor series of a certain function has integral coefficients, its anti-derivatives may all have fractional coefficients.

This does not work quite so well for our containers. Their series have sets as coefficients and sets do not come in fractional sizes. So we may not expect all analytic containers to have anti-derivatives.

Example 1.7.1: Let $F = (S \rhd P)$ where $S = 1$ and $P(*) = Fin(2)$. An anti-derivative would have some shape which has three positions since F has one with two positions. On the other hand, when you differentiate a shape with tree position you get at least three shape in the result, so F cannot be the derivative of such a container. Thus F has no anti-derivative.

There are many more examples of containers which lack anti-derivatives. Even our friend L, the list container, is among them. This might seem a bit depressing. We seem to lack some containers to fill the role of antiderivatives of common containers, and we even seem to lack some sets with fractional cardinality to fill their Taylor series. But we can think of it the other way around: If we find something to fill these roles, we will have a richer notion of a container, and thus have more structure to study.

Thus we set the programme for the rest of this thesis: To generalise the notion of a container - first only to see how far it may go - in the end to find a suitable level of generalisation where some interesting anti-derivatives live.

2 Categorified containers

In this section we define a categorified notion of a container.

We recall that a container is an object $S : Set$ and - looking at S as a discrete category - a functor $S \to Set$. It is irresistibly tempting to view Set as replaceable in that definition; the first thing which comes to mind is to replace it by Cat, the category of small categories and functors. So let us do just that.

Everything we describe in this part of the thesis is generalisation of notions from the case of containers. Thus, many of the proofs will be exactly parallel to the ones for containers, with only the addition of checking that it all works out for the arrows as well as the objects. Since this exercise in generalisation is undertaken mostly to provide a foundation for the third chapter, we will focus on the explicit constructions, and skip some of the tedious details which stem from the generalisation from sets to categories.

Although the theory in this chapter is quite general, it has some concrete applications. Especially the representation theorem (Theorem 2.2.1), which gives a combinatorial description of natural transformations of between certain endofunctors on Cat. This is illustrated in Example 2.2.2.

Before we go on to the definition, we will fix some notation to make our life a bit easier. Instead of writing $\sum_{a:A} B(a)$, we will write $\big(a : A;\ b : B(a)\big)$ with a semicolon. This reflects that the elements of this type are pairs (a, b) where $a : A$ and $b : B(a)$, and avoids subscripts.

2.1 Definition

We mechanically replace Set by Cat, functions by functors and dependent functions by natural transformations. What we obtain is the following:

Definition 2.1.1. The category of categorical containers, **CCon**, is the category defined by:

$S : Cat ; P : S \Rightarrow Cat$

- $Ob(CCon) = \big(S : Cat;\ P : Fun(S, Cat)\big) = \{(S \triangleright P)\}$.
- $Mor_{CCon}((S \triangleright P), (T \triangleright Q)) = \big(F : Fun(S, T);\ \sigma : Q \circ F \Rightarrow P\big)$.
- $(G, \tau) \circ (F, \sigma) = (G \circ F, \sigma \circ \tau F)$.
- $Id_{(S \triangleright P)} = (Id_S, Id_P)$

  An element $(S \triangleright P)$ in $Ob(CCon)$ is called a **categorical container**. Objects in S are called **shapes**, and given $s \in Ob(S)$, the objects of $P(s)$ are called **positions**.

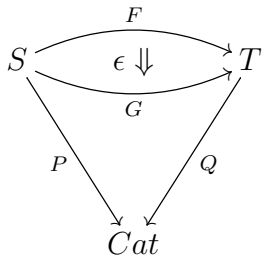
This is a direct generalisation of the container notion. I think it is practical to foreshadow some of the developments in the next chapter here, in order to motivate our next step, namely defining $CCon$ as a strict 2-category.

In the next chapter, in our search for anti-derivatives, it turns out that isomorphism of categorical containers is a very strict notion. In fact a bit too strict, we will see. From the above definition, we can see that there are now morphism between shapes, thus some shapes may be isomorphic.

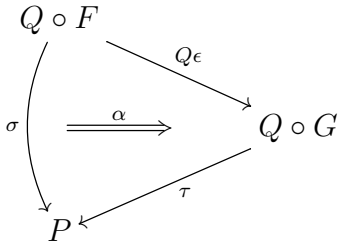
It is customary in category theory not to distinguish between isomorphic objects. But up to isomorphism of categorical containers, we can distinguish between a categorical container with a single shape and a categorical container with two isomorphic shapes, even if every mentioned shape has the same category of positions. So we seem to need a weaker notion of equivalence between categorical container.

One way of getting this sort of equivalence is to make $CCon$ into a strict 2-category. In every strict 2-category an equivalence between two objects A and B, is a pair of morphism $f : A \to B$ and $g : B \to A$, and 2-cells $\epsilon : f \circ g \cong id_B$ and $\delta : g \circ f \cong id_A$. In other words, it is a pair of morphism whose compositions need not be the identities, but merely isomorphic to the identities.

We should think for a second about what could be our 2-morphism between categorical container morphisms. Given a pair of morphism, (F, σ) and (G, τ), between two categorical containers, $(S \rhd P)$ and $(T \rhd Q)$, it feels natural that a 2-morphism should entail a natural transformation, ϵ, between F and G. Like this:



The natural transformations σ and τ are both morphisms into P, with their respective domains $Q \circ F$ and $Q \circ G$. We can push ϵ to fit between these two codomains using Q, and it seems in analogue to how 1-morphism are created to propose a natural transformation, $\alpha : \sigma \to \tau \circ Q\epsilon$ to be included in the 2-morphism.

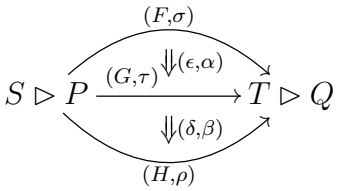


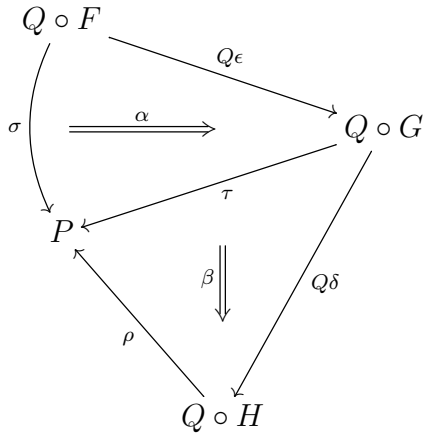
Definition 2.1.2. Given two categorical containers $(S \rhd P), (T \rhd Q) : CCon$ and two morphism $(F, \sigma), (G, \tau) : (S \rhd P) \to (T \rhd Q)$, a 2-morphism between (F, σ) and (G, τ) is a pair (ϵ, α), where $\epsilon : F \Rightarrow G$ and $\alpha : \sigma \Rightarrow \tau \circ Q\epsilon$.

We check that this does make Con into a strict 2-category.

- Vertical composition (where $\circ$ denote vertical composition of natural transformations):

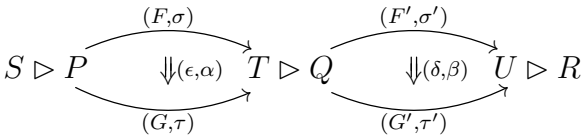
$$(\delta, \beta) \circ (\epsilon, \alpha) = (\delta \circ \epsilon, \beta(Q\epsilon) \circ \alpha)$$



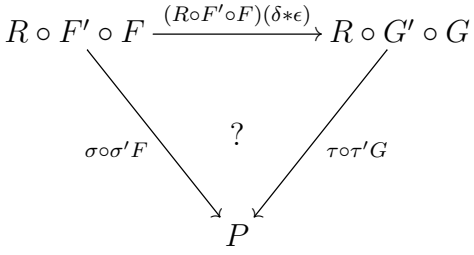


- Horizontal Composition:

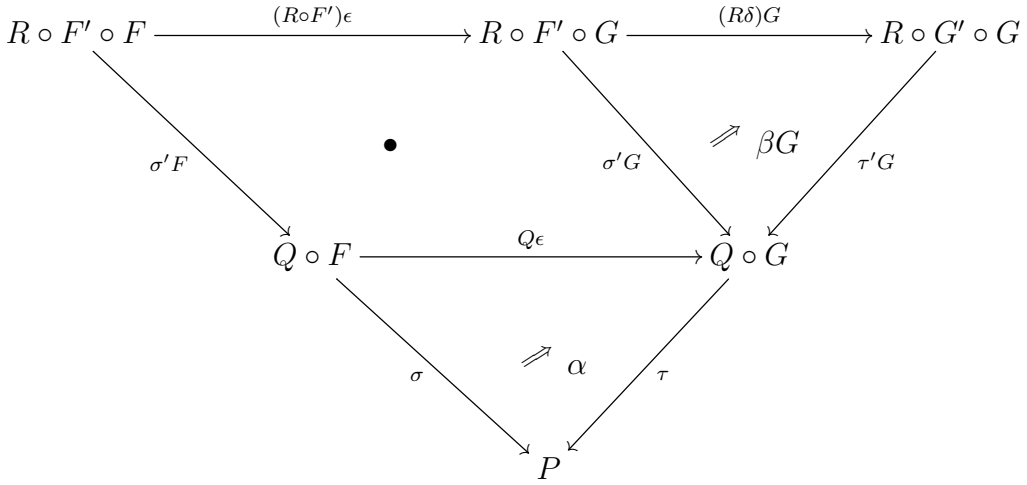
Given



and assuming that we want $\delta * \epsilon$ to be the first component of the composition, we are searching for a natural transformation to fill inn the diagram:



The previous diagram can be expanded into the following diagram:



This enables us to define:

$$(\delta, \beta) * (\epsilon, \alpha) : (F' \circ F, \sigma \circ \sigma' F) \Rightarrow (G' \circ G, \tau \circ \tau' G)$$
$$(\delta, \beta) * (\epsilon, \alpha) = (\delta * \epsilon, \tau(((R \circ F)\epsilon)(\beta G) \circ \alpha(\sigma' F))$$

We leave out the rest of the details of checking that this form a strict 2-category.

Actually there were two ways of defining the horizontal composition, since you can factor $R \circ F' \circ F \to R \circ G' \circ G$ in two ways. Either like we did above,

$$R \circ F' \circ F \to R \circ F' \circ G \to R \circ G' \circ G \tag{2.1.1}$$

or like this:

$$R \circ F' \circ F \to R \circ G' \circ F \to R \circ G' \circ G \tag{2.1.2}$$

But it turns out that these two definitions are the same.

To follow the setup in the previous chapter, we will now define a functor $[\![-]\!] : CCon \to Cat^{Cat}$, which extends the definition for Con. But categories have much more structure than sets, so writing out the definition will be tedious work. The reader should not be discouraged by this. The definition is a very natural extension of the original functor for containers. After the definition there is a set of diagrams, intended to illustrate where all the different morphisms belong.

Definition 2.1.3. There is a strict 2-functor $[\![-]\!] : CCon \to Cat^{Cat}$ defined by

- On objects $(S \rhd P) : Ob(CCon)$. $[\![S \rhd P]\!] : Cat \to Cat$ is the 2-functor defined by:
  - Given $C : Cat$, we let $[\![S \rhd P]\!](C)$ be the category defined by:
    - $Ob([\![S \rhd P]\!](C)) = \big(s \in Ob(S);\ \phi : P(s) \to C\big)$
    - $Mor_{[\![S \rhd P]\!](C)}((s,\phi),(t,\psi)) = \big(m \in Mor_S(s,t);\ \alpha : \phi \Rightarrow \psi \circ P(m)\big)$
    - $(n,\beta) \circ (m,\alpha) = (n \circ m, \beta P(m) \circ \alpha)$

    [Diagrams (2.1.3) left and right]
  - Given $C, D : Cat$ and a functor $F : C \to D$ let $[\![S \rhd P]\!](F) : [\![S \rhd P]\!](C) \to [\![S \rhd P]\!](D)$ be the functor defined by:
    - On objects: $[\![S \rhd P]\!](F)(s,\phi) = (s, F \circ \phi)$
    - On morphisms: $[\![S \rhd P]\!](F)(m,\alpha) = (m, F\alpha)$

    [Diagram (2.1.4) left]
  - Given $C, D : Cat$, functors $F, G : C \to D$ and a natural transformation $\epsilon : F \Rightarrow G$, we let $[\![S \rhd P]\!](\epsilon) : [\![S \rhd P]\!](F) \Rightarrow [\![S \rhd P]\!](G)$ be the natural transformation defined by:

    $$[\![S \rhd P]\!](\epsilon)_{(s,\phi)} : [\![S \rhd P]\!](F)(s,\phi) \to [\![S \rhd P]\!](G)(s,\phi)$$
    $$[\![S \rhd P]\!](\epsilon)_{(s,\phi)} = (id_s, \epsilon\phi)$$

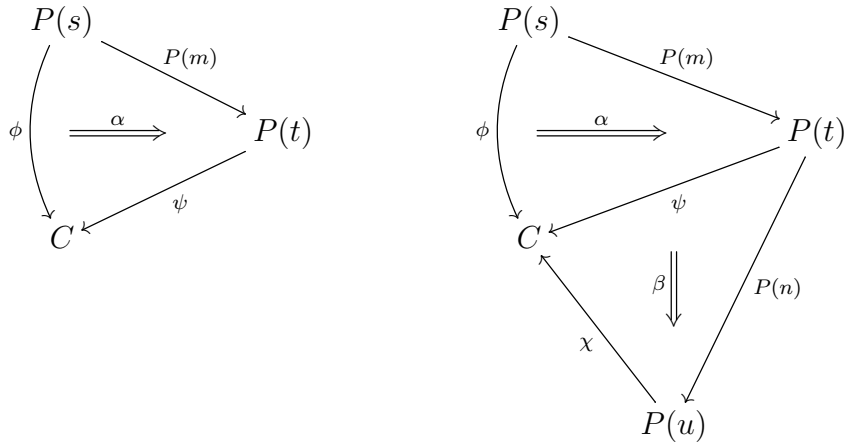
    [Diagram (2.1.4) right]
- On morphism $(F,\sigma) : S \rhd P \to T \rhd Q$ we let $[\![(F,\sigma)]\!] : [\![S \rhd P]\!] \Rightarrow [\![T \rhd Q]\!]$ be the natural transformation defined by:
  - $[\![(F,\sigma)]\!]_C : [\![S \rhd P]\!](C) \to [\![T \rhd Q]\!](C)$ is the functor defined by:
    - $[\![(F,\sigma)]\!]_C(s,\phi) = (F(s), \phi \circ \sigma_s)$
    - $[\![(F,\sigma)]\!]_C(m,\alpha) = (F(m), \alpha\sigma_s)$

    [Diagram (2.1.5)]
- On 2-morphisms $(\epsilon,\alpha) : (F,\sigma) \to (G,\tau)$, we let $[\![(\epsilon,\alpha)]\!] : [\![(F,\sigma)]\!] \Rightarrow [\![(G,\tau)]\!]$ be the natural transformation defined by:

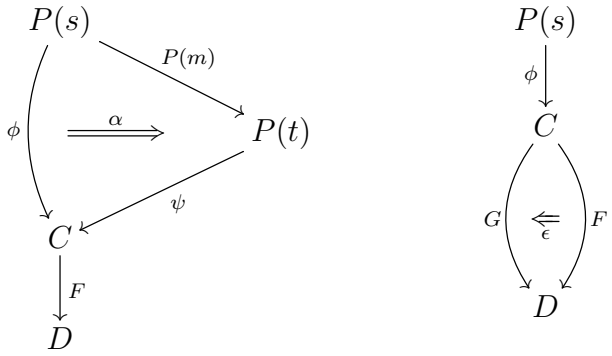
  $$[\![(\epsilon,\alpha)]\!]_{C,(s,\phi)} : [\![(F,\sigma)]\!]_C(s,\phi) \to [\![(G,\tau)]\!]_C(s,\phi)$$
  $$[\![(\epsilon,\alpha)]\!]_{C,(s,\phi)} = (\epsilon_s, \phi\alpha_s)$$

  [Diagram (2.1.6)]

Here are the diagrams illustrating the above definition:

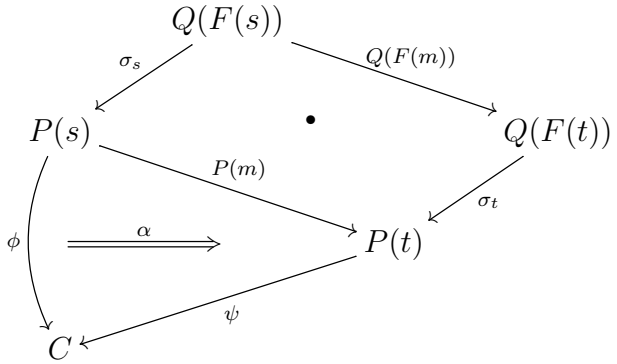


(2.1.3)

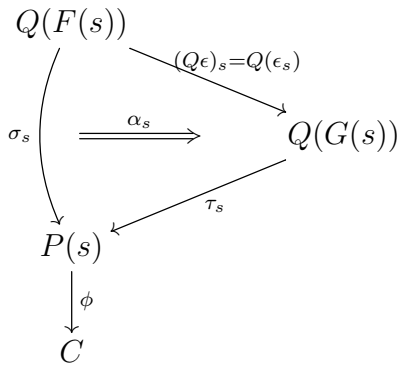


(2.1.4)

In the following diagram (2.1.5), we see that the top square is a naturality square for σ, and thus commutative.



(2.1.5)



(2.1.6)

Example 2.1.4: Let $\cdot \to \cdot$ denote the category with two objects, 0 and 1, with a single arrow $\mu : 0 \to 1$. Then we can define a categorical container $I^{\cdot \to \cdot}$, with the point 1_{Cat} as its category of shapes, whose shape $*$, has $\cdot \to \cdot$ as its category of positions.

Then given a category C, the category $[\![I^{\cdot \to \cdot}]\!](C)$ is simply the arrow category $C^{\cdot \to \cdot}$.

Example 2.1.5: The functor $[\![-]\!]$ preserves some interesting subcategories of Cat. Notably:

- If S is a set (discrete category), and $P(s)$ is a set for every $s \in S$, then $(S \rhd P)$ is actually a container, and $[\![S \rhd P]\!]$ can be restricted to a functor $Set \to Set$ which coincide with the one defined for containers.
- If S is a poset category, and the image of $P : S \to Cat$ is contained in the subcategory $Poset$ of poset categories, then it is clear from the definition of $[\![S \rhd P]\!]$ that it restricts to a functor $[\![S \rhd P]\!] : Poset \to Poset$.
- If S is a groupoid, and the image of $P : S \to Cat$ is contained $Grpd$, then $[\![S \rhd P]\!]$ restricts to a functor $[\![S \rhd P]\!] : Grpd \to Grpd$.

Example 2.1.6: If S is any small subcategory of Cat we can construct a categorical container $R(S) : CCon$. We define this by letting $R(S) = (S \rhd \iota)$ where ι is the inclusion of S into Cat.

This is a short cut to creating a some interesting categorical containers. For instance, let $Dia : Cat$ be some chosen skeletal subcategory of the category of finite categories. Then given $C : Cat$, we find that $[\![R(Dia)]\!](C)$ is the category of finite diagrams in C.

A lot of categorical containers are equivalent to one of the form $R(S)$ for some S. For instance all containers are of this kind, as well as all the categorical containers discussed so far. To see that not all categorical containers are of this kind: let S be a non-trivial group, and let P be the functor which collapses S onto the trivial group. This cannot be equivalent to $R(S)$, because the trivial group has no non-trivial automorphisms.

2.2 Properties

In this very general setting, with a 2-category of categorical containers, we will work out some typical constructions.

Representation

As we saw in the previous chapter, it was shown in (Michael Abbott et al. 2005) that, for containers, the functor $[\![-]\!] : Con \to Set^{Set}$ is fully faithful. A similar result is also true about our categorical containers. This is important because we want categorical containers to model the functors they represent as well as possible. This theorem says that all interaction between functors represented by categorical containers can be studied by studying their representations. The proof is parallel to the one for containers, only with more arrows.

Theorem 2.2.1. *The strict 2-functor $[\![-]\!] : CCon \to Cat^{Cat}$ is fully faithful. That is for any $(S \rhd P), (T \rhd Q) : CCon$, we have that $[\![-]\!]$ induces an isomorphism of categories $Mor_{CCon}(S \rhd P, T \rhd Q) \cong Mor_{Cat^{Cat}}([\![S \rhd P]\!], [\![T \rhd Q]\!])$.*

Proof. For every 2-cell $[\![S \rhd P]\!] \overset{F}{\underset{G}{\Downarrow \chi}} [\![T \rhd Q]\!]$ in Cat^{Cat}, we must construct a 2-cell $(S \rhd P) \overset{(f,\sigma)}{\underset{(g,\tau)}{\Downarrow (\epsilon,\alpha)}} (T \rhd Q)$ in $CCon$, such that $[\![-]\!]$ maps the second 2-cell to the first. The proof is analogue to the one for containers.

First we observe that $[\![S \rhd P]\!](1) \cong S$ and $[\![T \rhd Q]\!](1) \cong T$. So in Cat the cell

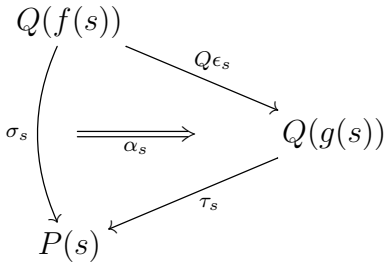
$$[\![S \rhd P]\!](1) \underset{G_1}{\overset{F_1}{\Downarrow \chi_1}} [\![T \rhd Q]\!](1)$$

is naturally isomorphic to a cell

$$S \underset{g}{\overset{f}{\Downarrow \epsilon}} T$$

for suitable $f, g : S \to T$ and $\epsilon : f \Rightarrow g$. So we let this define f,g and ϵ.

To define $\sigma : Q \circ f \Rightarrow P$, $\tau : Q \circ g \Rightarrow P$ and $\alpha : \sigma \Rightarrow \tau \circ Q\epsilon$ we first fix an $s \in Ob(S)$. Then we find components σ_s,τ_s and α_s to fill the diagram:



Unravelling the definition of $[\![S \rhd P]\!](P(s))$, we see that it has a very natural object $(s, id_{P(s)})$. From $[\![S \rhd P]\!](P(s))$ there are two functors $F_{P(s)}$ and $G_{P(s)}$ into $[\![S \rhd P]\!](P(s))$, and between them we have a natural transformation $\chi_{P(s)}$. So if we look at the image of $(s, id_{P(s)})$ under these two functor and the component of $\chi_{P(s)}$ between them we get:

$$F_{P(s)}(s, id_{P(s)}) \xrightarrow{\chi_{P(s),(s,id_{P(s)})}} G_{P(s)}(s, id_{P(s)}) \tag{2.2.1}$$

From the definition of $[\![S \rhd P]\!](P(s))$ we know that this is of the form:

$$(t, \phi) \xrightarrow{(m,\beta)} (t', \psi)$$

Where $t, t' \in T$, $\phi : Q(t) \to P(s)$, $\psi : Q(t') \to P(s)$, $m : t \to t'$ and $\beta : \phi \Rightarrow \psi \circ Q(m)$.

But because of naturality of F, G and χ, and how we defined f,g and ϵ, we know that (2.2.1) is actually

$$(f(s), \phi) \xrightarrow{(\epsilon_s, \beta)} (g(s), \psi)$$

for some $\phi : Q(f(s)) \to P(s)$, $\psi : Q(g(s)) \to P(s)$ and $\beta : \phi \Rightarrow \psi \circ Q(\epsilon_s)$. Thus, to complete our construction, we define:

$$\begin{aligned} \sigma_s &= \phi \\ \tau_s &= \psi \\ \alpha_s &= \beta \end{aligned}$$

□

The category $SCon$ is of a more combinatorial nature than Cat^{Cat}. One of the reasons for this is that $SCon$ does not involve any functors ranging over large categories, and is locally small. The following example shows how we can use this combinatoriality to characterise natural transformation between categorical container functors.

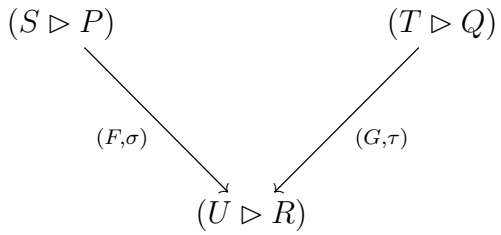
Example 2.2.2: Let $C : Cat \to Cat$ be the functor which maps a category X to the category of non-empty, finite chains in X. We consider the problem of characterising natural transformations $C \Rightarrow C$. This can be solved by noticing that C can be represented by a categorical container. Let $[n]$ define the poset category with $n+1$ objects $\{0, \cdots, n\}$, and a unique arrow from k to l if $k \leq l$. Then $[\![\mathbb{N} \rhd [-]]\!] \cong C$. So we know, by the representation theorem, that there is a bijective correspondence between $\sum_{f:\mathbb{N}\to\mathbb{N}} \prod_n ([f(n)] \to [n])$ and natural transformations $C \Rightarrow C$.

Sums and products

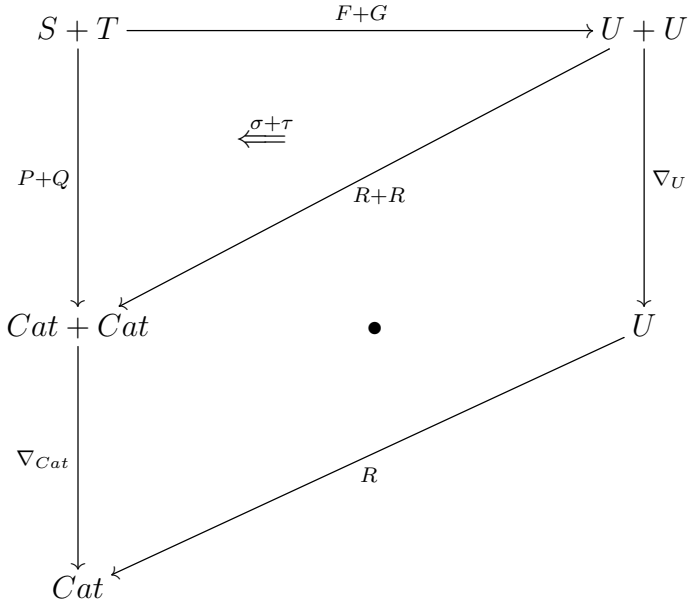
Sums and products in $CCon$ are analogous to their Con counterparts. Proving this can be done in two ways. Either by showing that the sum/product object has the universal property, or by using the fully faithful functor $[\![-]\!]$ and showing that the image of the sum/product is isomorphic to the product / coproduct in Cat^{Cat}. The later approach has the advantage of showing at the same time that $[\![-]\!]$ preserves products, but has the disadvantage of being more cumbersome. For the sake of variation we use the first strategy to prove existence of sums and the later for existence of products.

Theorem 2.2.3. *Given $(S \rhd P), (T \rhd Q) : CCon$ then the object $S + T = (S + T, \nabla_{Cat} \circ (P + Q))$, with the injections $(l_{S,T}, id_P)$ and $(r_{S,T}, id_Q)$,is the co-product of $(S \rhd P), T \rhd Q)$.*

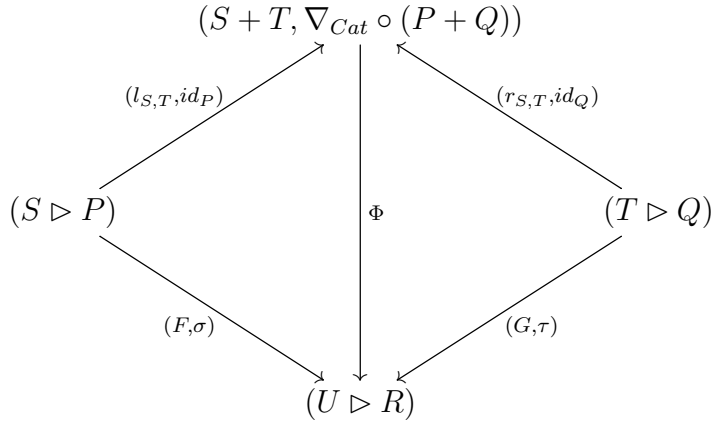
Proof. Let $(U \rhd R) : CCon$ and assume we have some morphism $(F, \sigma), (G, \tau)$:



Because of (2.2.2), we have that $\Phi = (\nabla_U \circ (F + G), \nabla_{Cat}(\sigma + \tau))$ is the unique morphism which makes (2.2.3) commute.



(2.2.2)



(2.2.3)

□

Theorem 2.2.4. *Given $(S \rhd P), (T \rhd Q) : CCon$ then their product is given by the object $(S \rhd P) \times (T \rhd Q) = (S \times T, P \circ \pi_0 + Q \circ \pi_1)$ and the projections $p_0 = (\pi_0, l_{P\circ\pi_0, Q\circ\pi_1})$ and $p_1 = (\pi_1, r_{P\circ\pi_0, Q\circ\pi_1})$.*

Proof. Because of the representation theorem it suffices to show that $[\![(S \rhd P) \times (T \rhd Q)]\!] \cong [\![(S \rhd P)]\!] \times [\![(T \rhd Q)]\!]$.

We have natural isomorphisms:

$$\begin{aligned}
Ob([\![(S \rhd P)]\!] \times [\![(T \rhd Q)]\!]\,(C)) &= (s \in S; C^{P(s)}) \times (t \in T; C^{Q(t)}) && (2.2.4)\\
&\cong ((s,t) \in S \times T; C^{P(s)} \times C^{Q(t)}) && (2.2.5)\\
&\cong ((s,t) \in S \times T; C^{P(s)+Q(t)}) && (2.2.6)\\
&\cong ((s,t) \in S \times T; C^{(P\circ\pi_0 + Q\circ\pi_1)(s,t)}) && (2.2.7)\\
&\cong Ob([\![(S \rhd P) \times (T \rhd Q)]\!]\,(C)) && (2.2.8)
\end{aligned}$$

Call the isomorphism from (2.2.4) to (2.2.8) f. Then we have:

$$\begin{aligned}
&Mor_{[\![(S\rhd P)]\!]\times[\![(T\rhd Q)]\!](C)}(((s,\phi),(s',\phi')),((t,\psi),(t',\psi')))\\
=&(m \in Mor_S(s,s'); \alpha : \phi' \circ P(m) \Rightarrow \phi)\\
&\times (n \in Mor_T(t,t'); \beta : \psi' \circ Q(n) \Rightarrow \psi)\\
\cong&((m,n) \in Mor_S(s,s') \times Mor_T(t,t')\\
&;(\alpha,\beta) \in (\phi' \circ P(m) \Rightarrow \phi) \times (\psi' \circ Q(n) \Rightarrow \psi))\\
\cong&((m,n) \in Mor_{S\times T}((s,t),(s',t'))\\
&;\gamma \in ((\phi' \circ P(m)) \times (\psi' \circ Q(n)) \Rightarrow \phi \times \psi))\\
\cong&((m,n) \in Mor_{S\times T}((s,t),(s',t'))\\
&;\gamma \in (((\phi' \times \psi') \circ (P(m) \times Q(n)) \Rightarrow \phi \times \psi))\\
\cong&(x \in Mor_{S\times T}((s,t),(s',t'))\\
&;\gamma \in (((\phi' \times \psi') \circ (((P \circ \pi_0) \times (Q \circ \pi_1))(x)) \Rightarrow \phi \times \psi))\\
\cong&(x \in Mor_{S\times T}((s,t),(s',t'))\\
&;\gamma \in (((\phi' \times \psi') \circ (((P \circ \pi_0) \times (Q \circ \pi_1))(x)) \Rightarrow \phi \times \psi))\\
=&Mor_{[\![(S\rhd P)\times(T\rhd Q)]\!](C)}(f((s,\phi),(s',\phi')),f((t,\psi),(t',\psi')))
\end{aligned}$$

□

Composition

Just like containers, categorified containers represent functors which can be composed. Thus it makes sense to ask if we can find a categorified container which represents this composition.

Definition 2.2.5. Given $(S \rhd P), (T \rhd Q) : CCon$ we define their **composition** $(S \rhd P)[(T \rhd Q)] = (\llbracket S \rhd Q \rrbracket (T) \rhd R)$ where $R : \llbracket S \rhd Q \rrbracket (T) \to Cat$ is given by:

$$Ob(R((s, f))) = (p \in Ob(P(s)); p' \in Ob(Q(f(p))))$$
$$Mor_{R((s,f))}(p, p', q, q') = (l : p \to q; k : (Q(f(m)))(p') \to q')$$

on objects, and on morphisms $(m, \alpha) : (s, \phi) \to (t, \psi)$ we have the function:

$$R(m, \alpha)(p, p') = (P(m)(p), (Q\alpha)(p'))$$
$$R(m, \alpha)(l, k) = (P(m)(l), (Q\alpha)(k))$$

This construction is parallel to how the composition of a container is defined, so we will not write out the proof that the object defined represents the composition.

2.3 Generalized differentiation?

Our motivation for looking at more general notions of containers was that we hoped to find the missing anti-derivatives. But so far we have not even defined differentiation of categorical containers.

The way differentiation was defined on decidable containers involved tearing out an arbitrary point from a set. Categories are quite different from decidable sets, they may have morphisms between their objects and just cutting out an object from a category would be evil.

To make this more precise: The object we cut out may be isomorphic to a another object, and if we leave this object in the category we end up with a category equivalent to the one we started with. And it would continue to be the same category up to equivalence until we have removed the whole isomorphism class. This process would necessarily distinguish between equivalent categories, which is the technical definition of evil.

Another problem is that we now have functors between our position categories, which needs to be reconstructed when we "remove an element" from the positions. What if one of these functors hit something which is removed?

In the next chapter we will begin by restricting to a subcategory of categorical containers where these issues are easily fixed.

3 Symmetric containers

In this chapter we restrict the notion of a categorical container to the notion of a symmetric container. This enables the definition of a differential operation on symmetric container, which we show has familiar properties such as distributivity over sums and Leibniz's rule for products. We find sufficient conditions on a symmetric container for constructions of an anti-derivative, and use this construction to find anti-derivatives for all analytic containers. This inspires some examples of differential equations of symmetric containers. Towards the end of this chapter we compare symmetric containers to combinatorial species and present some ideas for future study.

3.1 Definition and examples

In the previous chapter we described a very generalized notion of a container, and investigated how they represent endofunctors on the category of small categories. It was less clear if derivation makes sense for such a general definition.

The first problem was that removing objects from a category felt unnatural. This we will sidestep by just returning to discrete categories, i.e. sets. The second problem will be mediated by considering only categories of shapes which are groupoids.

This leads us to the definition:

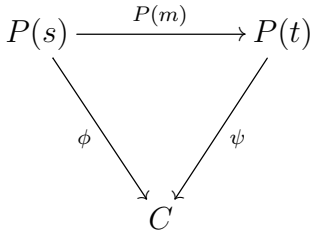
Definition 3.1.1. A **symmetric container** is a categorical container $(S \rhd P)$ such that S is a groupoid, and the image of P is contained in Set, considered as a full subcategory of Cat.

The category of symmetric containers, $SCon$, is the full subcategory of $CCon$ generated by symmetric containers.

Since $SCon$ is a subcategory of $CCon$, the construction $[\![-]\!]$ makes sense for symmetric containers as well. If $(S \rhd P) : SCon$, we observe that if we restrict the domain of $[\![S \rhd P]\!] : Cat \to Cat$ to $Grpd$, the codomain also restricts. Thus we have a meaningful restriction $[\![-]\!] : SCon \to Grpd^{Grpd}$.

Furthermore, if C is a set (discrete category), and given a morphism $(m, \alpha) : Mor_{[\![S \rhd P]\!](C)}((s, \phi), (t, \psi))$, α has to be the identity transformation, so that we really have:

$$Mor_{[\![S \rhd P]\!](C)}((s, \phi), (t, \psi)) = \{m : s \to t \mid \phi = \psi \circ P(m)\}$$



Compared to an arbitrary category, a groupoid is more like a topological space. We will freely use words like contractable, connected and component, when talking about groupoids. For a short introduction to groupoids, see the Appendix.

We will let $SCon$ inherit the strict 2-category structure from $CCon$ and use the accompanying notion of equivalence for comparing symmetric containers. An equivalence between two symmetric containers amount to an equivalence of the categories of shapes, with compatible isomorphisms of sets of positions.

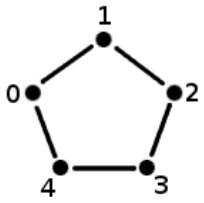
Inspection of the constructions in the previous chapter, shows that symmetric containers are closed under sums, products and composition.

To give some geometric intuition to the symmetric containers, we may sometimes sketch them. The convention will be that we draw the groupoid of shapes at the bottom of the sketch, and draw the sets of positions as dot clouds above their respective shapes. The action of the groupoid is indicated by lines between dots.

Observing that our symmetric containers are nothing more than groupoid actions on sets, we should have no shortage of examples.

Example 3.1.2: Any group has a natural action on its underlying set, defined by multiplication. A myriad of symmetric containers can be generated by sum from these containers.

Consider the cyclic group $\mathbb{Z}_5$. The symmetric container where $\mathbb{Z}_5$ acts on its underlying set is sketched in the Figure 3.1.5. Let us call this symmetric container F and let C_5 be given by $(\mathbb{Z}_5 \triangleright C_5) = F$.



$\bullet \quad \mathbb{Z}_5$

Figure 3.1.5: The symmetric container from the action of $\mathbb{Z}_5$ on its underlying set.

Given a set X, what does the groupoid $[\![F]\!](X)$ look like? According to the definition of $[\![-]\!]$, its objects are 5-tuples of elements in X, and a morphism between two such 5-tuples is an element of of $\mathbb{Z}_5$ whose action on one 5-tuple rotates it into the other. In relation to the drawing we just made, we can picture each of the dots in the pentagon decorated by an element from X, but being free to rotate the decorated pentagon using the $\mathbb{Z}_5$ action.

An interpretation of the groupoid $[\![F]\!](X)$ is that we have 5-tuples of elements of X but only partial information of about their relative positions. We know their order, but because of the $\mathbb{Z}_5$ action, we have no information about which one is first.

For example, the lack of information manifests that there is no natural transformation (natural in X) $[\![F]\!](X) \to X$, since there are no container morphisms $F \to I$. We can however get answers to questions of the kind: Are there two consecutive elements of the 5-tuple with a given decidable property? The word consecutive is used here in the sense that includes that first element is consecutive to the last element. Formally this manifest in an element of the end:

$$f \in \int_X \left(2^{X \times X} \to 2^{[\![F]\!](X)}\right)$$

$$f_X(p)(q) = \begin{cases} 1 & \exists m \in \mathbb{Z}_5 \; p(q(m), q(m+1)) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We see that if there is a morphism $k : q \to q'$ then $f_X(p)(q) = f_X(p)(q')$, so this is well defined. So intuitively we can answer this question because the answer is invariant under rotation.

In this example, we might as well have considered a general groupoid X. It would have made minimal difference.

Example 3.1.3: Let F be as in the previous example. Another group with a natural action on the set $\{0,1,2,3,4\}$ is the symmetry group S_5. Thus we let $G = (S_5 \triangleright P_5)$ be the corresponding symmetric container.

Following the lack-of-information interpretation of groupoids discussed in the previous example, $[\![G]\!](X)$ is the set of 5-tuples in X, where we know nothing of the relative positions of the tuple's elements.

Given an element of $[\![F]\!](X)$, we may forget the order of the 5-tuple, to obtain an element of $[\![G]\!](X)$. This stems from the inclusion $i : \mathbb{Z}_5 \to S_5$, and the natural transformation $\sigma : P_5 \circ i \Rightarrow C_5$ which is the identity of the set $\{0,1,2,3,4\}$. Together they form a morphism $(i,\sigma) : F \to G$. While there are many choices of $i : \mathbb{Z}_5 \to S_5$ and σ, the morphism $(i,\sigma) : F \to G$ is unique up to isomorphism of morphisms.

Example 3.1.4: Let $\mathbb{Z}_* = \sum_{n\in\mathbb{N}_+} \mathbb{Z}_n$ be the co-product of all the finite cyclic groups as a groupoid. For simplicity we let $Ob(\mathbb{Z}_*) = \mathbb{N}_+$. There is an action $C : \mathbb{Z}_* \to Set$ defined by

$$\begin{aligned} &C(n) : Set \\ &C(n) = Mor_{\mathbb{Z}_*}(n,n) = \{0, \cdots, n-1\} \end{aligned}$$

on objects, and on morphisms $k : Mor_{\mathbb{Z}_n}(*,*)$:

$$\begin{aligned} &C(k) : P(n) \to P(n) \\ &C(k) = \lambda l.(k \circ l) = \lambda l.(k + l \mod n) \end{aligned}$$

We then have a symmetric container $(\mathbb{Z}_* \triangleright C)$.

Example 3.1.5: Let S_n be the n-th symmetric group. That is, $S_n = Aut(Fin(n))$, the set of isomorphism $Fin(n) \to Fin(n)$. And let $S_* = \sum_{n\in\mathbb{N}} S_n$. For simplicity we let $Ob(S_*) = \mathbb{N}$. We have a natural action $\mathcal{P} : S_* \to Set$ defined by objects by

$$\begin{aligned} &\mathcal{P}(n) : Set \\ &\mathcal{P}(n) = Fin(n) \end{aligned}$$

and on morphisms $\sigma : Mor_{S_n}(*,*)$ by

$$\mathcal{P}(\sigma) : Fin(n) \to Fin(n)$$
$$\mathcal{P}(\sigma) = \lambda x.\sigma(x)$$

We then have a symmetric container $(S_* \rhd \mathcal{P})$. Given X, we can think about $[\![S_* \rhd \mathcal{P}]\!](X)$ as all finite sequences with elements in X but where we do not know anything about their order. In that respect we can see them as multisets.

3.2 Differentiation

Now we can define what it means to differentiate a symmetric container. The definition is a generalization of the one for containers and makes use of the translation groupoid construction $B(S, P)$ (See the Appendix).

Definition 3.2.1. A symmetric container $(S \rhd P)$ is **decidable** if $P(s)$ is a decidable set for every $s \in Ob(S)$.

Definition 3.2.2. Let $F = (S \rhd P)$ be a decidable symmetric container. Its **derivative** $\partial F = (B(S, P) \rhd P')$, were:

$$P'(s, p) = P(s) - p$$
$$P'(m : (s, p) \to (t, q)) = P(m)|_{p,q}$$

Example 3.2.3: Let us differentiate $(\mathbb{Z}_4 \rhd C_4)$, the symmetric container where $\mathbb{Z}_4$ acts on it self. The translation groupoid $B(\mathbb{Z}_4, C_4)$ has four objects, namely $(*, 0)$,$(*, 1)$,$(*, 2)$ and $(*, 3)$. A morphism from $(*, x)$ to $(*, y)$ is a morphism k in $\mathbb{Z}_4$ such that $x + k = y \mod 4$. There is a unique morphism from any object to any other object, and the sets of positions each have three elements.

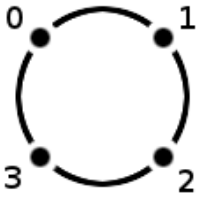


Figure 3.2.6: The symmetric container from the action of $\mathbb{Z}_4$ on its underlying set.

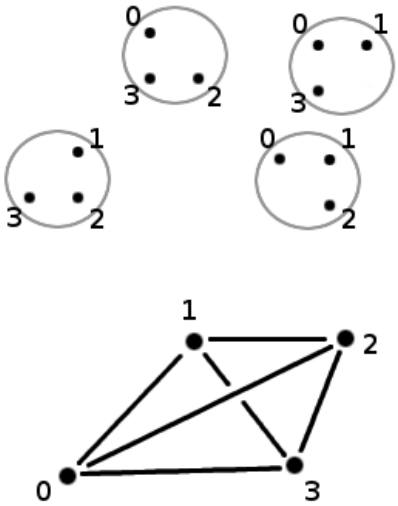


Figure 3.2.7: The symmetric container $\partial(\mathbb{Z}_4 \rhd C_4)$.

Differentiation of symmetric containers, like differentiation of containers, is not functorial. But it preserves both isomorphism and equivalence of symmetric containers. This is not surprising, but it would be very disturbing if it was not true.

Proposition 3.2.4. *If* $(S \rhd P) \simeq (T \rhd Q)$*, then* $\partial(S \rhd P) \simeq \partial(T \rhd Q)$*. Furthermore, if* $(S \rhd P) \cong (T \rhd Q)$*, then* $\partial(S \rhd P) \cong \partial(T \rhd Q)$*.*

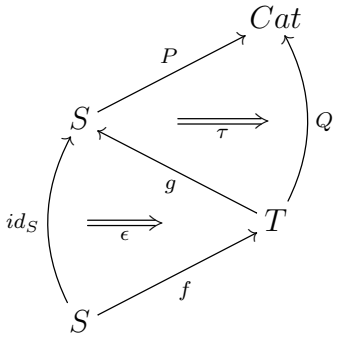
Proof. We only prove that differentiation preserves equivalence. The proof that it preserves isomorphism is similar but easier.

An equivalence $(S \rhd P) \simeq (T \rhd Q)$ contains the following data: $(f, \sigma) : (S \rhd P) \to (T \rhd Q)$, $(g, \tau) : (T \rhd Q) \to (S \rhd P)$, $(\epsilon, \alpha) : id_{(S \rhd P)} \overset{\cong}{\to} (g, \tau) \circ (f, \sigma)$ and $(\delta, \beta) : id_{(T \rhd Q)} \overset{\cong}{\to} (f, \sigma) \circ (g, \tau)$.

So what we need to construct is:

$$
\begin{aligned}
&f' : B(S, P) \to B(T, Q) \\
&\sigma' : Q' \circ f' \Rightarrow P' \\
&g' : B(T, Q) \to B(S, P) \\
&\tau' : P' \circ g' \Rightarrow Q' \\
&\epsilon' : id_{S'} \overset{\cong}{\to} g' \circ f' \\
&\alpha' : id_{P'} \overset{\cong}{\to} (\sigma' \circ (\tau' f')) \circ P' \epsilon' \\
&\delta' : id_{T'} \overset{\cong}{\to} f' \circ g' \\
&\beta' : id_{Q'} \overset{\cong}{\to} (\tau' \circ (\sigma' g')) \circ Q' \delta'
\end{aligned}
$$

Starting with f', the diagram,



enables us to define it like this:

$$
\begin{aligned}
f'(s, p) &= (f(s), (\tau f \circ P\epsilon)_s(p)) \\
f'(m) &= f(m)
\end{aligned}
$$

Naturality of $\tau f \circ P\epsilon$ ensures that if m is in $Mor_{B(T,Q)}((s, p), (t, q))$ then $f(m)$ is in $Mor_{B(T,Q)}(f'(s, p), f'(t, q))$.

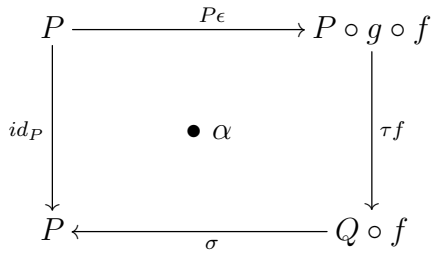
Unravelling the definition of derivation, we see that $\sigma'_{(s,p)}$ should a be function:

$$\sigma'_{(s,p)} : Q(f(s)) - (\tau f \circ P\epsilon)_s(p) \to P(s) - p$$

Thus, if we can prove that $\sigma \circ \tau f \circ P\epsilon = id_P$, we can let σ' be defined by

$$\sigma'_{(s,p)} = \sigma|_{(\tau f \circ P\epsilon)_s(p),p}$$

Actually we have a proof of that at hand, namely α. A bit of unravelling of the definition of a 2-morphism, gives:



While α is only a natural isomorphism, we know that the diagram commutes since all fibres of P are discrete.

We can construct g' and τ' in a similar fashion:

$$\begin{aligned} g'(t,q) &= (g(t), (\sigma g \circ Q\delta)_t(q)) \\ g'(m) &= g(m) \end{aligned}$$

$$\tau'_{(t,q)} = \tau|_{(\sigma g \circ Q\delta)_t(q),q}$$

To finish this off, we can see that $(g' \circ f')(s,p) = (g(f(s)), P\epsilon_s(p))$, so we may let $\epsilon'_{(s,p)} = \epsilon_s$. By carefully observing that all the points cut in the definitions match up, we can construct α' as well as $\alpha_{(s,p),q} = \alpha_{s,q}$. The construction of δ' and β' is dual. □

3.3 Differentiation of sums and products

We now turn to proving that the familiar rules from calculus which tell us how to differentiate sums and products are also true for our symmetric containers.

Proposition 3.3.1. *For symmetric containers F and G, we have*

$$\partial(F + G) \cong \partial F + \partial G$$

Proof. Let $F = (S \triangleright P)$ and $G = (T \triangleright Q)$. And let $(B(S+T, \nabla_{Cat} \circ (P+Q)) \triangleright R') = \partial(F+G)$ and $(B(S,P) + B(T,Q), \nabla_{Cat}(P' + Q') = \partial P + \partial Q$, where P' and Q' are the ones given by the definition of ∂F and ∂G.

First we construct an isomorphism $f : B(S+T, \nabla_{Cat} \circ (P+Q)) \to B(S,P) + B(T,Q)$.

On objects:

$$f((l_{S,T}(s), p)) = l_{B(S,P),B(T,Q)}(s,p)$$
$$f((r_{S,T}(t), q)) = r_{B(S,P),B(T,Q)}(t,q)$$

On morphisms

$$f(l_{S,T}(m)) = l_{B(S,P),B(T,Q)}(m)$$
$$f(r_{S,T}(m)) = r_{B(S,P),B(T,Q)}(m)$$

since we can define f^{-1} by reversing the above equations, f is an isomorphism. Given $(l_{S,T}(s), p) \in Ob(B(S+T, \nabla_{Cat} \circ (P+Q)))$, we observe:

$$\begin{aligned} R'(l_{S,T}(s), p) &= \nabla_{Cat}((P+Q)(l_{S,T}(s))) - p \\ &= P(s) - p \end{aligned}$$

Furthermore:

$$\begin{aligned} \nabla_{Cat}(P' + Q')(f(l_{S,T}(s), p)) &= \nabla_{Cat}(P' + Q')(l_{B(S,P),B(T,Q)}(s,p)) \\ &= P'(s,p) \\ &= P(s) - p \end{aligned}$$

Thus we can let $\sigma_{(l_{S,T}(s),p)}$ be the identity on $P(s) - p$. Dually we can let $\sigma_{(r_{S,T}(t),q)}$ be the identity on $Q(t) - q$. Thus we can conclude that $(f, \sigma) : \partial(F+G) \to \partial F + \partial G$ is an isomorphism.

□

Proposition 3.3.2. *Leibniz's rule For symmetric containers F and G, we have*

$$\partial(F \times G) \cong \partial F \times G + F \times \partial G$$

Proof.

$$f : B(S \times T, (+) \circ (P \times Q)) \to B(S,P) \times T + S \times B(T,Q)$$
$$f((s,t), l_{P(s),Q(t)}(p)) = l_{B(S,P)\times T, S \times B(T,Q)}((s,p),t)$$
$$f((s,t), r_{P(s),Q(t)}(q)) = r_{B(S,P)\times T, S \times B(T,Q)}(s,(t,q))$$

We see that f is an isomorphism. A similar argument to the one in the proof of Prop 3.3.1 shows that there is also a σ such that (f, σ) becomes an isomorphism.

□

3.4 Composition and the chain rule

Remember that if we had two categorical containers, F and G, we could find their composition $F[G]$. Inspecting the construction, we see that if F and G are symmetric containers, their composition is also a symmetric container.

Proposition 3.4.1. *The chain rule For symmetric containers F and G, we have*

$$\partial(F[G]) \cong \partial F[G] \times \partial G$$

Proof. Let $F = (S \rhd P)$ and $G = (T \rhd Q)$. We construct the isomorphism $(f, \sigma) : \partial(F[G]) \to \partial F[G] \times \partial G$

Objects in the groupoid of shapes for $\partial(F[G])$ are of the form $((s, \phi), (p, q))$ where

$$\begin{aligned} &s : S \\ &\phi : P(s) \to T \\ &p : P(s) \\ &q : Q(f(p)) \end{aligned}$$

On the other hand, the groupoid of shapes for $\partial F[G] \times \partial G$ has objects of the form $(((s, p), \psi), (t, q))$ where

$$\begin{aligned} &s : S \\ &p : P(s) \\ &\psi : P(s) - p \to T \\ &t : T \\ &q : Q(t) \end{aligned}$$

So given a shape $((s, \phi), (p, q))$ in $\partial(F[G])$, we can define

$$f((s, \phi), (p, q)) = (((s, p), \phi|_{P(s)-p}), (\phi(p), q))$$

We leave out the rest of the details, as they introduce nothing new. □

3.5 Anti-derivatives

Our motivating example is L. The container of lists. Viewed as a container in the non-symmetric sense, it has no anti-derivative. For instance it contains only one

shape with three positions. Any decidable anti-derivative would have to have at least one shape with four positions. But by differentiation, this shape would give rise to not only one, but four shapes with five positions.

This is reflected in the fact that $\int x^3 \; dx = \frac{1}{4}x^4 + C$, and we cannot have $\frac{1}{4}$ of a shape with four positions. But if we allow groupoids, it would seem that we can. Following the definition of cardinality of a groupoid (see the Appendix), $\mathbb{Z}_4$ has cardinality $\frac{1}{4}$. Looking at the derivative of the symmetric container where $\mathbb{Z}_4$ acts on the set of four elements, which we calculated in an earlier example; it is clear that it is *equivalent* to a single shape with three positions. Maybe we can find some symmetric container which is the anti-derivative of L, not up to isomorphism, but up to equivalence of symmetric containers.

To search for an anti-derivative for L, we could begin by integrating its Taylor-series:

$$\int (\sum_{n:\mathbb{N}} x^n) \; dx = \sum_{n:\mathbb{N}_+} \frac{1}{n} x^n + D$$

From this we see that we need a container with $\frac{1}{n}$ shapes with n elements for all n. The obvious candidate for an anti-derivative for L is $(\mathbb{Z}_* \rhd C)$, the cyclic container. At least it has the correct number of shapes. So what happens when we differentiate it?

The translation groupoid of C consists of a countably infinite number of contractable components, namely the simplices (see figure below). Thus it is equivalent to $\mathbb{N}$ as a groupoid. In each n-simplex component, the shapes have exactly n positions. We conclude that we have that $\partial(\mathbb{Z}_* \rhd C)$ is equivalent to L as a symmetric container.

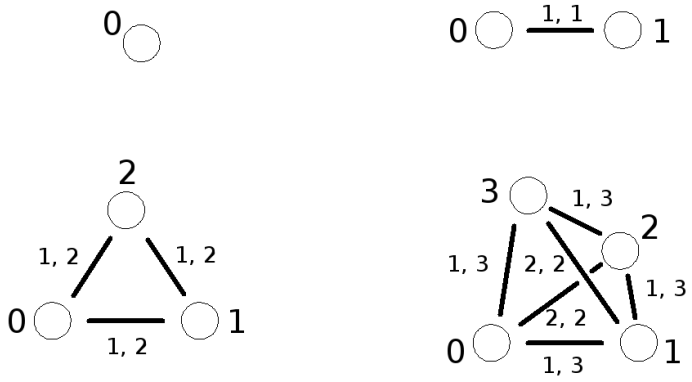


Figure 3.5.8: The first four components of $B(Z_*, C)$, the groupoid of shapes in $\partial(\mathbb{Z}_* \rhd C)$. (Identity morphisms not included.)

Definition 3.5.1. Given two symmetric containers $F, G : SCon$, we say that G is an **anti-derivative** of F if $F \simeq \partial G$.

Example 3.5.2: In calculus, anti-derivatives are unique up to a constant. But this is not true for symmetric containers. Consider the two containers $\mathbb{Z}_4 \rhd C_4$ and $\mathbb{Z}_2 \times \mathbb{Z}_2 \rhd P$, where $P : \mathbb{Z}_2 \times \mathbb{Z}_2 \to Set$ is the canonical action of $\mathbb{Z}_2 \times \mathbb{Z}_2$ on it self by addition. These are clearly not equivalent since $\mathbb{Z}_4 \not\simeq \mathbb{Z}_2 \times \mathbb{Z}_2$. But if you differentiate them, they will both become equivalent to I^3.

On the other hand, if two decidable symmetric containers differ only by a constant, i.e. $F \simeq G + K(A)$, then they are anti-derivatives of the same container: $\partial F \simeq \partial G$.

These examples of anti-derivatives suggest a strategy to construct anti-derivatives for symmetric containers with discrete groupoids of shapes (i.e. containers). For each shape, if we could select a group structure on the set of positions of that shape with an extra position added, we could use the coproduct (as groupoids) of those groups as our new groupoid of shapes. We then get a symmetric container where the groups act on them selves. When differentiated, this new symmetric container would have one contractable component for each of the shapes of the original container. It would not be difficult to see that it is an anti-derivative of the original container.

If we employed the axiom of choice to select group structures, the above reasoning would produce a theorem saying that all containers have an anti-derivative among the symmetric containers. But invoking the axiom of choice to create a container makes it less useful for combinatorial or computational purposes. Instead we will focus on generalising the procedure to other symmetric containers, keeping the assumption that necessary structures are provided. To do this we will need to stitch together groups indexed over a groupoid in a uniform way. We achieve this by using the tools of internal logic in categories. In particular we use the idea of an internal group object (See the Appendix) and a tiny bit of topos theory. For a reference on logical topos theory, see (Rupert Goldblatt 1984) or (Joachim Lambek and Scott 1986).

Let $(S \rhd P)$ be a symmetric container and consider the topos Set^S, where P is an object. In Set^S there is also an object $P + 1$, constructed by adding an element to every set of positions in P. A topos also has enough structure so that we can talk about group objects, and in particular the internal automorphism group of $P + 1$, which we will denote $Aut(P + 1)$.

The following theorem is a bit surprising as it gives sufficient conditions for the existence of an anti-derivative for a general container which at first sight seem to have little to do with differentiation.

Theorem 3.5.3. *Let $(S \rhd P) : SCon$ be a decidable symmetric container. Given a group object $\Psi : Set^S$, and an internal group action $\psi : \Psi \to Aut(P + 1)$ such that*

the morphism $(\lambda x.\psi(x)(r(*))) : \Psi \to P + 1$ *is an isomorphism, we may construct a symmetric container* $(\tilde{S} \rhd \tilde{P})$ *which is an anti-derivative of* $(S \rhd P)$.

Proof. Notice that, because Ψ in an internal group in Set^S, the fibres $\Psi(s)$ are groups. We will denote by e_s the identity in $\Psi(s)$ and juxtaposition will denote multiplication.

We define $\tilde{S} : Grpd$ by:

$$\begin{aligned} Ob(\tilde{S}) &= Ob(S) \\ Mor_{\tilde{S}}(s,t) &= Mor_S(s,t) \times \Psi(s) \\ (m,g) \circ (n,h) &= (m \circ n, \Psi(m^{-1})(g)h) \end{aligned}$$

(Functoriality of Ψ shows that multiplication is associative.)

We define $\tilde{P} : \tilde{S} \to Set$ by:

$$\tilde{P}(s) = P(s) + 1$$

and given $(m,g) : Mor_{\tilde{S}}(s,t)$:

$$\begin{aligned} \tilde{P}(m,g) &: (P+1)(s) \to (P+1)(t) \\ \tilde{P}(m,g) &= (P+1)(m) \circ \psi_s(g) \end{aligned}$$

To show that we have found an anti-derivative, we need to construct an equivalence $(S \rhd P) \simeq \partial(\tilde{S} \rhd \tilde{P})$.

$$\begin{aligned} f &: S \to B(\tilde{S}, \tilde{P}) \\ f(s) &= (s, r(*)) \\ f(m) &= (m, e_s) \end{aligned}$$

$$\begin{aligned} g &: B(\tilde{S}, \tilde{P}) \to S \\ g(s,p) &= s \\ g(m,g) &= m \end{aligned}$$

Let $\phi : P + 1 \to \Psi$ be the inverse of $(\lambda x.\psi(x)(r(*))) : \Psi \to P + 1$.

$$\begin{aligned} \epsilon &: Id_{\tilde{S}'} \Rightarrow f \circ g \\ \epsilon_{(s,p)} &: (s,p) \to (s, r(*)) \\ \epsilon_{(s,p)} &= (id_s, \phi_s(p)) \end{aligned}$$

Which has the inverse:

$$\epsilon^{-1} : f \circ g \Rightarrow Id_{\tilde{S}'}$$
$$\epsilon^{-1}_{(s,p)} : (s,p) \to (s, r(*))$$
$$\epsilon^{-1}_{(s,p)} = (id_s, \phi_s(x)^{-1})$$

On the other hand, we see that $g \circ f$ actually is id_S, so it is the easy direction of the equivalence.

Creating an invertible natural transformation $\sigma : \tilde{P}' \circ f \Rightarrow P$ is also straight forward since $\tilde{P}'(f(s)) = P(s) + 1 - r(*)$ which is naturally isomorphic to $P(s)$.

Similarily for $\tau : P \circ g \Rightarrow \tilde{P}'(s,p)$ we have that $P(g(s,p)) = P(s)$ and $\tilde{P}'(s,p) = P(s) + 1 - p$, which is isomorphic to $P(s)$ if $P(s)$ is decidable.

□

Example 3.5.4: Let us test the theorem on a simple test case. Let $F = (\mathbb{Z}_2 \triangleright C_2)$ be the cyclic action of $\mathbb{Z}_2$ on it self. The topos we are to look at is $Set^{\mathbb{Z}_2}$, the category of $\mathbb{Z}_2$-sets. Let $\Psi : Set^{\mathbb{Z}_2}$ be given by:

On objects:

$$\Psi(*) = \{0, 1, 2\}$$

On morphisms, generated by 1 in $\mathbb{Z}_2$:

$$\Psi(1) : \{0,1,2\} \to \{0,1,2\}$$
$$\Psi(1)(x) = x + 1 \mod 3$$

So Ψ is the $\mathbb{Z}_2$-set, $\{0,1,2\}$ where $\mathbb{Z}_2$ acts by swapping 1 and 2. Consider $C_2 + 1$ which is the $\mathbb{Z}_2$-set $\{l(0), l(1), r(*)\}$ where $\mathbb{Z}_2$ acts by swapping $l(0)$ and $l(1)$. There is a group action $\psi : \Psi \to Aut(C_2 + 1)$ defined by:

$$\psi_*(1)(l(0)) = l(1)$$
$$\psi_*(1)(l(1)) = r(*)$$
$$\psi_*(1)(r(*)) = l(0)$$

We see that $\lambda x.\psi(x)(r(*))$ is an isomorphism as required. Its inverse $\phi : C_2 + 1 \to \Psi$ is given by:

$$\phi_*(r(*)) = 0$$
$$\phi_*(l(0)) = 1$$
$$\phi_*(l(1)) = 2$$

Since the definition of $\tilde{S}$ in the theorem preserves the set of objects, the groupoid of shapes we get in this case will be a group. We see that the group will have six elements. The question is thus which six element group will we get? There are only two options, either $\mathbb{Z}_6$ or S_3.

The underlying set is $\{0,1\} \times \{0,1,2\}$, so at first we might think that it is obviously $\mathbb{Z}_2 \times \mathbb{Z}_3 \cong \mathbb{Z}_6$. But looking at the way the multiplication is defined might complicate the matter:

$$(a, x) \circ (b, y) = (a + b, \Psi(a^{-1})(x) + y)$$

A simple way to distinguish between $\mathbb{Z}_6$ and S_3 is to see how many idempotent elements there are. Not counting the identity, $\mathbb{Z}_6$ has one, while S_3 has two. In fact both $(1,0)$ and $(1,1)$ are idempotent:

$$\begin{aligned}(1,0) \circ (1,0) &= (1+1, \Psi(0)(0) + 0) \\ &= 0 \\ (1,1) \circ (1,1) &= (1+1, \Psi(1)(1) + 1) \\ &= (0, 2+1) \\ &= (0,0)\end{aligned}$$

From this we conclude that the resulting symmetric container has S_3 as its groupoid of shapes. To decide how an element (a, x) of this group acts on the set of positions $\{0,1\} + 1$ we can see from the definition that it first uses $\psi_*(x)$ to rotate all three elements around, then the action of a to swap $l(0)$ and $l(1)$ iff $a = 1$. We know that this generates the usual permutation action of S_3 on a set with three elements.

In retrospect we should praise that it was not $\mathbb{Z}_6$ which came out of this construction, as it could not possibly have created an anti-derivative of our container. It could only have acted on the three positions by rotation, so that if one element was fixed, every element would be fixed, making it impossible for $\mathbb{Z}_2$ to show up.

This theorem has a satisfying corollary, which is that every analytic container has an anti-derivative.

Corollary 3.5.5. *Let $(S \rhd P)$ be an analytic container, considered as a symmetric container. Then $(S \rhd P)$ has an anti-derivative.*

Proof. Remember that given a series of sets $A : \mathbb{N} \to Set$, we constructed $C(A)$, which was a container. An analytic container F was one that was isomorphic to $C(T(F))$. Thus it clearly suffices to show that $C(A)$ has an anti-derivative for any A.

Let $(S \rhd P) = C(A)$. From the definition of $C(A)$ we know that $S = \sum_{n \in \mathbb{N}} A(n)$, we can therefore define $\Psi : Set^S$ by:

$$\Psi(x, n) = \{0, \cdots, n\}$$

Addition modulo $n+1$ in each fibre makes Ψ a group object. From the definition of $C(A)$ we see that $P(x, n) \cong \{0, \cdots, n-1\}$, thus $P(x, n) + 1 \cong \{0, \cdots, n-1, n\}$, so we may define ψ by:

$$\begin{aligned} &\psi_{(x,n)} : \{0, \cdots, n\} \to Aut(\{0, \cdots, n\}) \\ &\psi_{(x,n)}(k)(i) = k + i \mod n + 1 \end{aligned}$$

□

3.6 Differential equations

Having found some anti-derivatives, it might be fun to look at some examples of differential equations.

Example 3.6.1: Let $F = (S_* \rhd \mathcal{P})$ be the previously discussed symmetric container. It is the solution to the following differential equation, with the initial condition $[\![F]\!](0) = 1$

$$\partial F \simeq F$$

We can see this by observing that $B(S_{n+1}, \mathcal{P}_{n+1}) \simeq S_n$, since the subgroup of S_{n+1} which fixes a given position is all permutations of the other positions, of which there are n.

Given a groupoid X, with cardinality $|X| = x$, the cardinality

$$\begin{aligned} |\,[\![F]\!](X)| &= \sum_{n \in Ob(S_*)} \frac{|X|^n}{|Mor_{S_n}(*, *)|} \\ &= \sum_{n \in \mathbb{N}} \frac{x^n}{n!} = e^x \end{aligned}$$

Hence F plays the role as the exponential function!

Example 3.6.2:
Consider the following differential equation of symmetric containers.

$$\partial F \simeq F \times F$$

In other words, removing a position from this kind structure would give us an pair of structures of the same kind. Adding an initial condition, we can attempt to solve this equation by iteratively making pairs and anti-differentiating the result to generate new shapes. For instance, the initial condition $[\![F]\!](0) = 1$, and calculation of the first three iterations gives the symmetric container sketched below.

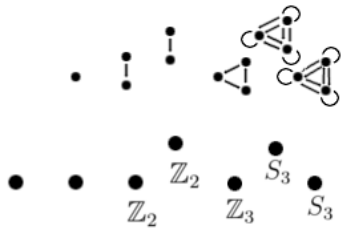


Figure 3.6.9: The first components of a possible solution to the differential $\partial F \simeq F \times F$, given $[\![F]\!](0) = 1$.

There is also another solution to this differential equation, namely the list container L.

3.7 Relation to species

The notion of a symmetric container is closely related to the notion of a species of structure. But while symmetric containers operate on all sets, species operate only on finite sets. In the following we let $\mathbb{E}$ be the category of finite sets and functions, and $\mathbb{B}$ be the category of finite sets and bijections.

Definition 3.7.1. A **species of structure** is a functor $F : \mathbb{B} \to \mathbb{E}$. An element $s \in F(X)$ is called an F**-structure on** X.

Definition 3.7.2. Given a species of structure F and a finite set X, there is an associated groupoid $\widetilde{F}(X)$, defined as follows.

$$Ob(\widetilde{F}(X)) = F(X)$$
$$Mor_{\widetilde{F}(X)}(s_0, s_1) = \{\sigma : X \xrightarrow{\cong} X \mid F(\sigma)(s_0) = s_1\}$$

Given a species of structure F, we can define a groupoid,

$$S_F = \sum_{n \in \mathbb{N}} \widetilde{F}(Fin(n)),$$

and a functor $P : S \to Set$,

$$P_F(n, s) = Fin(n)$$
$$P_F(\sigma) = \sigma.$$

This defines for each species of structure a symmetric container $(S_F \rhd P_F)$. Since symmetric species can have infinite numbers of positions, there can be no going back.

3.8 Future work

Let us allow our self a moment of speculation about what other things which could be investigated in relation to the subject of symmetric containers.

First of all it would be interesting look at symmetric containers where the positions are from a different category than Set. If replace Set by Vec_k, the category of vector spaces over a field k, we could perhaps use the tools of representation theory. On vector spaces with interior products, differentiation could give a groupoid version of the projective spaces acting on subspaces of codimension 1.

It might also be interesting to try to create a formal language with a differential operator on types. Maybe with a Haskell-like GADT syntax (General Algebraic Data Type), so that for instance a representation of $F = (S_* \rhd \mathcal{P})$ would be defined as:

```
data F a where
   empty : F a
   sym   : F a -> dF a
```

This definition reflects the differential equation and initial conditions satisfied by F. With a canonical projection `p : a -> dF a -> F a`, you could construct elements of `F a` by recursion. For instance you could represent the forgetful morphism from lists to F by:

```
forget : [a] -> F a
forget [] = empty
forget (x : xs) = p x (sym (forget xs))
```

The cyclic container $(\mathbb{Z}_* \triangleright C)$, being an anti-derivative of the list container, could be represented by:

```
data C a where
   e    : dC a
   cons : a -> dC a -> dC a
```

Pattern matching could be used on the projection `p : a -> dF a -> F a`, given that all the variables matched are encapsulated in `p` again. Like in the following recursive definition:

```
f : C a -> F a
f (p x e) = p x (sym empty)
f (p x (cons y ys)) = p x (sym (f (p y xs)))
```

Such a language could be useful for manipulating data types with symmetries. An application would be to create a programming language where some data type invariants are enforced on type-level, without introducing dependent types.

Appendix

Groupoids

While the name "groupoid" correctly suggests that groupoids are generalised groups, we see them more as special categories where every morphism is an isomorphism. Consequently we see groups as specialised groupoids with only one object.

An easy way to make more complicated groupoids from groups is taking their coproduct. Coproducts in the category of groupoids is more like the coproduct in Set than the free product of groups. Given two groupoids G and H, their coproduct $G + H$ has as its objects the disjoint union of the objects in G and the objects in H. The morphisms in each component are simply the ones inherited from G and H and we add no morphism between.

This construction can be extended easily to countable coproducts. For instance we can take some well known families of groups and make them into groupoids:

$$\mathbb{Z}_* = \sum_{n:\mathbb{N}_+} \mathbb{Z}_n$$
$$S_* = \sum_{n:\mathbb{N}} S_n$$

For simplicity of notation we let $Ob(\mathbb{Z}_*) = \mathbb{N}_+$ and $Ob(S_*) = \mathbb{N}$.

An action of a groupoid G in some category $\mathcal{C}$ is simply a functor $G \to \mathcal{C}$. S_n is defined by its action on $Fin(n)$, so there is a canonical permutation action $P : S_* \to Set$ defined by:

$$P(n) = Fin(n)$$
$$P(\sigma)(x) = \sigma(x)$$

To provide a similar action for $\mathbb{Z}_*$ we need to pick a bijection $|\mathbb{Z}_n| \cong Fin(n)$. So by recursion we define bijections $\phi : \forall n \in \mathbb{N}_+ \; Fin(n) \to |\mathbb{Z}_n|$ by:

$$\phi_1(*) = 0$$
$$\phi_{n+1}(l(x)) = \phi_n(x)$$
$$\phi_{n+1}(r(*)) = n$$

This bijection lets us think of $Fin(n)$ as the set $\{0, 1, \cdots, n\}$.

And so we can define an action $C : \mathbb{Z}_* \to Set$ by:

$$C(n : \mathbb{N}) = Fin(n)$$
$$C(x : |\mathbb{Z}_n|)(a : Fin(n)) = \phi^{-1}(\phi(a) + x)$$

In other words C cycles the elements of $Fin(n)$ in a particular order, while P allow any permutation.

A useful construction is the translation groupoid of a groupoid action on sets. Given any groupoid action $P : G \to Set$ we can form the translation groupoid $B(G, P)$:

- $Ob(B(G, P)) = (s : Ob(G); P(s))$
- $Mor_{B(G,P)}((s, p), (t, q)) = \{g : Mor_G(s, t) \mid P(g)(p) = q\}$
- Composition as inherited from G.

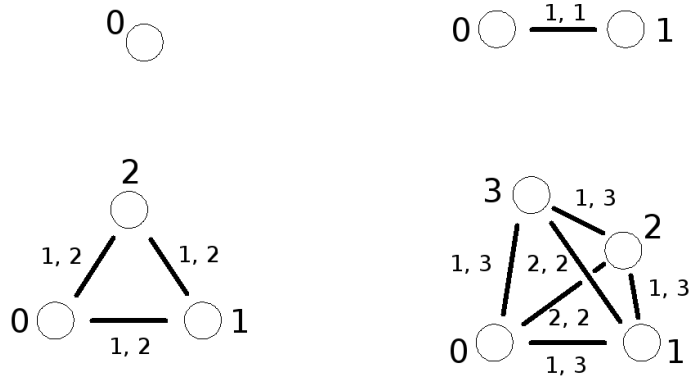


Figure A.1: The first four components of $B(\mathbb{Z}_*, C)$. Each line represents a pair of isomorphisms going oposite in directions. Identity morphisms excluded.

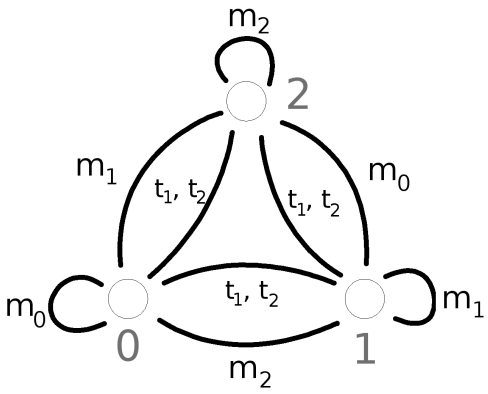


Figure A.2: The translation groupoid of S_3 acting on $Fin(3)$. The elements of S_3 are the identity (not shown), the translations t_1 and t_2 and the reflections m_0, m_1 and m_2.

Cardinality of a groupoid

There is a notion of cardinality of a groupoid, where groupoids have a positive real number as its cardinality. Given a groupoid G, its **cardinality** is defined to be

$$|G| \cong \sum_{[x] \in Ob(G)/\cong} \frac{1}{|Mor_G(x,x)|}$$

Where $[x] \in Ob(G)/\cong$ choses an object from each isomorphism class of objects, and $|Mor_G(x,x)|$ is the cardinality of the set of morphisms from x ot itself. If the sum diverges we say that G has infinite cardinality.

This notion is based on classical mathematics, as it involves choice and cardinality of sets. In this thesis we only use this notion on groupoids where it is natural to chose a skeleton and the sets of morphisms are finite and decidable.

The definition of cardinality of groupoids can be found in (John Baez, Alexander Hoffnung and Christopher Walker 2009), along with other groupoidifications.

Group object in a category

In category theory, one can take structures which are normally defined in terms of sets and functions, and describe them in terms of objects and morphisms in order to look for similar structures in other categories. This has for instance been done for groups.

Definition 3.8.1. Given a category $\mathcal{C}$ with products and a terminal object 1, an **internal group** is an object G along with morphisms $m : G \times G \to G$, $e : 1 \to G$ and $i : G \to G$, called **multiplication**, **identity** and **inverse**, such that the following diagrams commute:

Associativity:

$$\begin{array}{ccc} G \times G \times G & \xrightarrow{id_G \times m} & G \times G \\ {\scriptstyle m \times id_G}\downarrow & & \downarrow{\scriptstyle m} \\ G \times G & \xrightarrow[m]{} & G \end{array}$$

Identity ($id_G \times e$ is short for $(id_G \times (e \circ \square_G)) \circ \Delta_G$):

$$\begin{array}{ccc} G & \xrightarrow{id_G \times e} & G \times G \\ {\scriptstyle e \times id_G}\downarrow & \searrow^{id_G} & \downarrow{\scriptstyle m} \\ G \times G & \xrightarrow[m]{} & G \end{array}$$

Inverse:

$$\begin{array}{ccccc} G & & \xrightarrow{(id_G \times i) \circ \Delta_G} & & G \times G \\ & \searrow^{\square_G} & & & \\ {\scriptstyle (i \times id_G) \circ \Delta_G}\downarrow & & 1 & & \downarrow{\scriptstyle m} \\ & & & \searrow^{e} & \\ G \times G & & \xrightarrow[m]{} & & G \end{array}$$

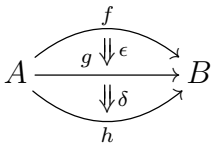
In *Set* an internal group is just a group. In *Top*, the category of topological spaces, an internal group is a topological group. A Lie-group is a group internal to *Diff*, the category of smooth manifolds and smooth functions.

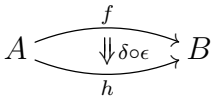
2-categories

A 2-category is a like a category, but between two objects, instead of a set of morphism, we have a small category of morphisms. Further more, composition should be functorial. A 2-category is *weak* if associativity and the other identities which hold in a category only hold up to isomorphism, and *strict* if they holds up to equality. We will work with strict 2-categories in this thesis. For reference on 2-categories, see (Saunders Mac Lane 1998).

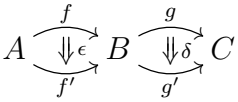
In 2-categories we have both horisontal and vertical composition. We will denote vertical composition by $\circ$ and horizontal composition by $*$.

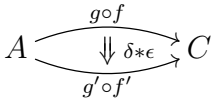
Vertical composition:





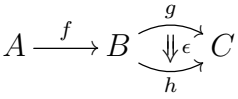
Horizontal composition:

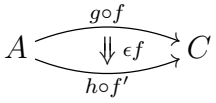




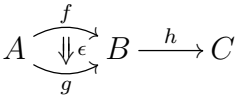
Whiskering of 2-cells is denoted by juxtaposition.

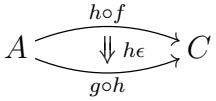
Right whiskering:





Left whiskering:





Haskell code

The following code is a demonstration of how differentiation can be used to generate trees. We saw in the first chapter how we could calculate the number of trees by differentiation. But if we refrain from using the associativity and commutativity of the cartesian product, we get the actual structures of the trees. It is however horribly inefficient as it generates each tree $n!$ times, where n is the number of leaf nodes. This is because differentiation disciminiates between different orderings of leaf nodes. The code may be improved by creating more specialised rules for addition and multiplication which simplifies the expressions in the right way.

```
-- Define binary trees
import Data.List

data BinTree = L | B BinTree BinTree
deriving (Eq, Show, Ord)

-- Define a tree generator
data TreeGenerator = Zero        -- No trees
                   | One         -- Filled
                   | X           -- Leaf node
                   | Y           -- Recursion stopper
               -- Branching:
                   | Plus TreeGenerator TreeGenerator
                   | Mult TreeGenerator TreeGenerator
deriving (Eq, Show)


-- Plus wrapper:
plus Zero a = a
plus (Mult One a) (Mult One b) = mult One (plus a b)
plus a b = Plus a b

-- Multiplication wrapper:
mult Zero _ = Zero
mult Y (Plus a b) = plus a (Mult Y b)
mult a b = Mult a b

-- Generate a set of trees from TreeGenerator
generate :: TreeGenerator -> [BinTree]
```

```
generate Zero = []
generate One = return L
generate X = []
generate Y = []
generate (Plus l r) = generate l ++ generate r
generate (Mult l r) = do
x <- generate l
y <- generate r
return (B x y)

-- Differentiate a tree generator:
differentiate :: TreeGenerator -> TreeGenerator
differentiate Zero = Zero
differentiate One = Zero
differentiate X = One
differentiate Y = Zero
differentiate (Mult Y r) = mult Y (differentiate r)
differentiate (Plus l r) =
    plus (differentiate l) (differentiate r)
differentiate (Mult l r) =
    plus (mult l (differentiate r)) (mult (differentiate l) r)


-- Strict Binary Trees:
t = Plus X (Mult Y (Mult t t))


--- Get a list of all strict binary trees, grouped by size.
trees = map (unique . sort . generate) $ iterate differentiate t


unique [] = []
unique (a:[]) = [a]
unique (a:b:xs) = if (a == b) then ur else a:ur where
     ur = unique (b:xs)
```

For example you can list all strict binary trees with up to five leaf nodes by running the following code in GHCi (The Glasgow Haskell Compiler interactive mode):

```
 > take 5 trees
[[],[L],[B L L],[B L (B L L),B (B L L) L],
[B L (B L (B L L)),B L (B (B L L) L),B (B L L) (B L L),
B (B L (B L L)) L,B (B (B L L) L) L]]
```

The output is a list of five lists. Each list contains all strict binary trees with a certain number of leaf nodes.

References

Bergeron, Labelle and Leroux: 1998, *Combinatorial species and tree-like structures*, Vol. 67 of *Encyclopedia of Mathematics and its Applications*, Cambridge University Press, Cambridge.

Joachim Lambek and Scott, P. J.: 1986, *Introduction to higher order categorical logic*, Vol. 7 of *Cambridge Studies in Advanced Mathematics*, Cambridge University Press, Cambridge.

John Baez, Alexander Hoffnung and Christopher Walker: 2009, Higher-dimensional algebra VII: Groupoidification.

Martin Hofmann and Streicher, T.: 1998, The groupoid interpretation of type theory, *Twenty-five years of constructive type theory (Venice, 1995)*, Vol. 36 of *Oxford Logic Guides*, Oxford Univ. Press, New York, pp. 83–111.

Michael Abbott, Thorsten Altenkirch and Neil Ghani: 2005, Containers: constructing strictly positive types, *Theoretical Computer Science* **342**(1), 3–27.

Michael Abbott, Thorsten Altenkirch, Neil Ghani and Conor Mcbride: 2008, ∂ for data: Differentiating data structures.

Philip Wadler: 1990, Comprehending monads, *1990 ACM Conference on Lisp and Functional Programming*, ACM, ACM Press, pp. 61–78.

Rupert Goldblatt: 1984, *Topoi*, Vol. 98 of *Studies in Logic and the Foundations of Mathematics*, second edn, North-Holland Publishing Co., Amsterdam. The categorial analysis of logic.

Saunders Mac Lane: 1998, *Categories for the working mathematician*, Vol. 5 of *Graduate Texts in Mathematics*, second edn, Springer-Verlag, New York.

Simon Byrne: 2005, On groupoids and stuff.
URL: *http://www.math.mq.edu.au/ street/ByrneHons.pdf*

Thomas Streicher: 2006, *Domain-theoretic foundations of functional programming*, World Scientific.

Thorsten Altenkirch, Paul Levy and Staton, S.: 2010, Higher-order containers, *in* F. Ferreira, B. Löwe, E. Mayordomo and L. M. Gomes (eds), *Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE 2010, Ponta Delgada, Azores, Portugal, June 30 - July 4, 2010. Proceedings*, Vol. 6158 of *Lecture Notes in Computer Science*, Springer, pp. 11–20.
URL: *http://dx.doi.org/10.1007/978-3-642-13962-8*