Acta Informatica manuscript No.
(will be inserted by the editor)

∂ for Data

Differentiating Data Structures

Michael Abbott[1], Thorsten Altenkirch[2],
Neil Ghani[1], Conor McBride[3]

[1] Department of Mathematics and Computer Science, University of Leicester.
e-mail: `michael@araneidae.co.uk`, e-mail: `ng13@mcs.le.ac.uk`
[2] School of Computer Science and Information Technology, Nottingham University.
e-mail: `txa@cs.nott.ac.uk`
[3] Department of Computer Science, University of Durham.
e-mail: `c.t.mcbride@durham.ac.uk`

The date of receipt and acceptance will be inserted by the editor

Abstract. This paper and our conference paper (Abbott, Altenkirch, Ghani, and McBride, 2003b) explain and analyse the notion of the derivative of a data structure as the *type of its one-hole contexts* based on the central observation made in McBride (2001). To make the idea precise we need a generic notion of a data type, which leads to the notion of a container, introduced in Abbott, Altenkirch, and Ghani (2003a) and investigated extensively in Abbott (2003). Using containers we can provide a notion of linear map which is the concept missing from McBride's first analysis. We verify the usual laws of differential calculus including the chain rule and establish laws for initial algebras and terminal coalgebras.

1 Introduction

This paper, based on our conference paper (Abbott et al., 2003b) explains and analyses the notion of the derivative of a data structure as the *type of its one-hole contexts* based on the central observation made in McBride (2001).

One-hole contexts arise frequently in symbolic programming tasks such as the implementation of a tactic library for a proof system such as COQ or LEGO. For a simple example, consider the task of writing editing operations for binary trees (in Haskell):

```
data Tree = Leaf | Node Tree Tree
```

We often require the notion of a tree with a hole replacing of one of its subtrees. How can we represent this in a functional language? A good answer to this question can be found in Huet's functional pearl, '*The Zipper*' (Huet, 1997).

Using a top-down approach for ease of presentation[1], we arrive at the following Haskell program:

```
data Tree' = Left Tree | Right Tree
type Zipper = [Tree']
```

Here `Tree'` represents the choice we have to make at every step on the path through the tree together with the rest of the tree not on our path. The `Zipper` is the list of these steps. This can be made precise by providing a `plug` operation which fills a hole with a tree:

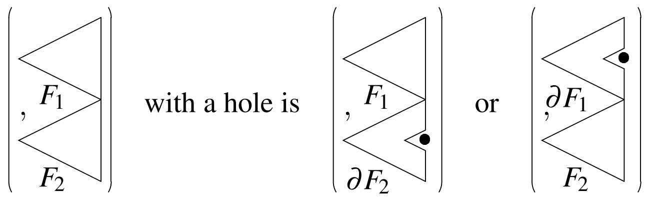
```
plug :: Zipper -> Tree -> Tree
plug [] t = t
plug ((Left l) : z) t = Node (plug z t) l
plug ((Right r) : z) t = Node r (plug z t)
```

Can we solve this problem in a generic way? What happens if we consider ternary trees or finitely branching trees? Certainly the `Zipper` will remain a sequence of steps but what are these steps made of? Writing $T = \mu X.F\,X$ for the generic type of trees, the zipper arises as $Z = \mathsf{List}\,(F'\,T)$, where $\mathsf{List}\,X = \mu Y.1 + X \times Y$ and F' is derived from F in some way. In the case of binary trees we have $F\,X = 1 + X^2$ and $F'\,X = 2 \times X$. In the case of ternary trees we have to remember where we go and two subtrees, i.e. $F\,X = 1 + X^3$ leads to $F'\,X = 3 \times X^2$.

It turns out that the resemblance with derivatives is no accident and applies to all reputable type constructors. E.g. what is the type of one-hole contexts of $F\,X = F_1\,X \times F_2\,X$? A hole in F is either a hole in F_1 leaving F_2 intact or vice versa a hole in F_2 with F_1 intact. Writing ∂F for F with a hole we arrive at the product law

$$\partial F \cong \partial F_1 \times F_2 + F_1 \times \partial F_2 \quad .$$

We might picture this as follows:



[1] Huet's slightly more complex bottom-up approach yields more efficient operations, and is based on the same data structures

In the present paper we cover all ingredients of polynomial functors, the chain rule and rules for initial algebras and terminal coalgebra not present in calculus. An example for the initial algebra case is the derivative of lists $\partial \mathsf{List}\, X = \mu Z.\mathsf{List}\, X + X \times Z$. A similar construction can be applied to potentially infinite lists $\mathsf{List}^\infty X = \nu Y.1 + X \times Y$ whose derivative is $\partial \mathsf{List}^\infty X = \mu Z.\mathsf{List}^\infty X + X \times Z$. Note that the μ in the derivative does not change into a ν which reflects the fact that the path to a hole is always finite.

To make the idea precise we need a generic notion of a data type, which leads to the notion of a container, introduced in Abbott et al. (2003a) and investigated extensively in Abbott (2003). A container is a generic notion of a datatype, it is a functor of the form of a formal power series:

$$F\, X = \Sigma s\!:\!S.P\, s \to X$$

we call S the set of shapes and P the family of positions. These are closely related to analytical functors (Joyal, 1986; Hasegawa, 2002) and generalise shapely types (Jay and Cockett, 1994). In Hoogendijk and de Moor (2000) containers are investigated in a relational setting.

Using containers we can provide a notion of linear map which is the concept missing from McBride's first analysis (McBride, 2001). We define a linear morphism between containers as a polymorphic function which does not copy or forget data, this corresponds to a cartesian natural transformation on the associated functors. Writing $\mathbf{Con}^{\multimap}(F, G)$ for the set of linear maps we can specify derivatives by the following isomorphism

$$\mathbf{Con}^{\multimap}(F \times \mathsf{I}, G) \cong \mathbf{Con}^{\multimap}(F, \partial G)$$

We can then construct the derivative of a decidable container, i.e. one where the set of positions has a decidable equality, explicitely as

$$\begin{aligned} F\, X &= \Sigma s\!:\!S.P\, s \to X \\ \partial F\, X &= \Sigma s\!:\!S.\, \Sigma p\!:\!P\, s.\, (P\, s - p) \to X \end{aligned}$$

where $A - a$ is the set A without the element a. This clearly resembles derivatives of a power series:

$$\begin{aligned} f\, x &= \textstyle\sum_{i=0}^{\infty} x^{a_i} \\ \partial f\, x &= \textstyle\sum_{i=1}^{\infty} a_i x^{a_i - 1} \end{aligned}$$

In Abbott et al. (2003b) we used a the language of category theory to present our construction in a point-free way, while in the present paper uses the language of Martin-Löf's constructive set theory. In a way nothing has changed because the constructive set theory is just the internal language of the class of categories we are interested in. However, in our experience it is easier to keep track of the dependencies within constructions by using variables. Using the type-theoretic language we arrive at a picture where our proofs very closely resembles the intuitive reasoning illustrated by diagrams and hence explains better how the proofs were actually discovered.

1.1 Related work

Ehrhard and Regnier (2001) introduce the differential lambda calculus, however their motivation is quite different from ours in that they use differentiation on programs not types. However, they use the notion of a linear substitution to define differentiation of which our $\mathbf{Con}^{\multimap}$ may be an instance.

Fiore, Plotkin, and Turi (1999) introduce the operator δ on functor categories by the isomorphism

$$F \times \mathsf{I} \to G \cong F \to \delta G$$

similar to our specification of ∂ but this operator does not behave like differentiation because the morphisms are not restricted to the linear (i.e. cartesian) natural transformations.

1.2 Plan of the paper

In section 2 we review the constructive set theory we are using and relate it to a class of categories. We also introduce the pattern matching notation which is used throughout the paper. In section 3 we revisit the notion of containers, based on material in Abbott et al. (2003a) and Abbott (2003). In section 4 we introduce the notion of subtraction and establish a number of properties which we need later. We also define the notion of a decidable container. In section 5 we specify the notion of a derivative using linear maps and show how to construct the derivative of a decidable container explicitely. In section 6 we apply this construction to establish a number of laws, i.e. how to construct derivatives of constant containers, $+$,$\times$ and the chain rule. In section 7 we show how to construct least and greatest fixpoints of containers and in section 8 we introduce and verify laws for the differentiation of fixpoints. We close with conclusions and suggestions for further work.

2 Apparatus

This paper can be read in two ways:

1. as a construction within Martin-Löf's constructive set theory;

2. as a construction in the internal language of a class of categories.

The set theory we are using is the extensional theory $\mathbf{MLW^{ext}}$, e.g. see Aczel (1999), and is defined by the following constructions:

Π-sets: We shall use the notation $(a:A) \to B\,a$ for Π-sets and the usual λ-calculus syntax for abstraction and application.

Σ-sets: We write Σ-sets as $(a:A;\ B\,a)$, its elements are written (a,b), we use pattern matching notation to implement elimination. We notationally extend this to n-ary Σ-sets by

$$(a_1:A_1;\ a_2:A_2;\ \ldots\ a_n:A_n) := (a_1:A_1;\ (a_2:A_2;\ \ldots\ a_n:A_n)\ldots)\ .$$

We also write 1 for a set with just one constructor $():1$.

Coproducts: It is sufficient to introduce $\mathsf{Bool}:\mathrm{Set}$ with $\mathsf{true},\mathsf{false}:\mathsf{Bool}$ and $0:\mathrm{Set}$ with no constructors. Formally, binary coproducts can be reduced to Bool via

$$A+B = (b:\mathsf{Bool};\ \mathsf{if}\,b\,\mathsf{then}\,A\ \mathsf{else}\,B)\ .$$

Using a vertical pattern matching notation, inspired by functional programming languages like SML or Haskell, we write

$$A+B = \begin{pmatrix} \ \ \mathsf{true}\ ;\ A \\ \mid \mathsf{false};\ B \end{pmatrix}\ .$$

Here we use pattern matching to calculate a set, this is called a *large elimination*. In the sequel we will use inl and inr as the constructors for $A+B$ and again use pattern matching notation for elimination. In the presence of large eliminations we can also define sets by pattern matching, e.g. in the case of Σ-sets we write

$$\begin{pmatrix} \ \ \mathsf{inl}\,a{:}A\,;\ C\,a \\ \mid \mathsf{inr}\,b{:}B;\ D\,b \end{pmatrix}$$

for $(x{:}A+B;\ F\,x)$ where

$$F\,(\mathsf{inl}\,a) = C\,a$$
$$F\,(\mathsf{inr}\,b) = D\,b\ .$$

In Abbott et al. (2003a), Abbott (2003) and Abbott et al. (2003b) we used the point-free notation $(A+B;\ C \mathbin{\overset{\circ}{+}} D)$ to denote this set. In this paper we prefer the use of pattern matching notation over point-free notation, in order to show precisely the dependencies which we exploit.

Equality sets: Given $x,y:X$, we have $\mathsf{Eq}\,x\,y:\mathrm{Set}$. It contains the single element $\mathsf{refl}\,x$ if and only if $x=y$, and is otherwise uninhabited. As we are working in an extensional setting, we may treat x and y as the same, if we know that $\mathsf{Eq}\,x\,y$ is inhabited. In this view it is easy to see that

Eq is symmetric and transitive and we can establish that constructors are one-to-one:

$$\begin{aligned}\mathsf{Eq}\,(\mathsf{inl}\,a)\,(\mathsf{inl}\,a') &\cong \mathsf{Eq}\,a\,a' \\ \mathsf{Eq}\,(\mathsf{inr}\,b)\,(\mathsf{inr}\,b') &\cong \mathsf{Eq}\,a\,a' \\ \mathsf{Eq}\,(\mathsf{inl}\,a)\,(\mathsf{inr}\,b) &\cong 0 \\ \mathsf{Eq}\,(\mathsf{inr}\,b)\,(\mathsf{inl}\,a) &\cong 0 \\ \mathsf{Eq}\,(a_1,c_1)\,(a_2,c_2) &\cong (\mathsf{Eq}\,a_1\,a_2;\ \mathsf{Eq}\,c_1\,c_2)\ .\end{aligned}$$

W-sets: Given $A:\mathrm{Set}, B:A \to \mathrm{Set}$ we write $\mathsf{W}\,A\,B:\mathrm{Set}$ for the inductive set generated by $\sup:(a:A) \to (B\,a \to \mathsf{W}\,A\,B) \to \mathsf{W}\,A\,B$. We can use function definition by structural recursion as an elimination principle. Dually we may introduce the coinductive counterpart of W which we denote by $\mathsf{M}\,A\,B$ allowing guarded corecursion as elimination. However, in our forthcoming paper Abbott, Altenkirch, and Ghani (2004b), we show that M-sets can be constructed from W-sets in this theory.

We do not assume the presence of a universe of small sets.

We will sometimes name subterms which correspond to places in a diagram, even if the names play no part in the formal mathematics. Hence the reader should not be surprised to encounter terms like:

$$\begin{array}{lll}(a{:}A)+(b{:}B) & \text{for} & A+B \\ (s{:}S;\ p{:}P\,s) & \text{for} & (s{:}S;\ P\,s)\end{array}$$

Categorically, our assumptions correspond to our ambient category of sets having the following properties:

- locally cartesian closed (LCC), this corresponds to having Σ-sets, 1, Π-sets and equality sets;
- disjoint coproducts, corresponding to the presence of large eliminations for Bool (and hence all coproducts);
- W-sets correspond to assuming that initial algebras for functors of the form $F\,X = (a{:}A;\ B\,a \to X)$ exist, and dually M-types correspond to terminal coalgebras for the same class of functors.

In this framework a set X is interpreted as an object of the ambient category, while a family $X \to \mathrm{Set}$ is interpreted as an object of the slice category over X. All the type theoretic constructions then lift to constructions on the slice categories.

In Abbott, Altenkirch, and Ghani (2004a) we have referred to locally cartesian closed categories with disjoint coproducts and W-sets as **Martin-Löf categories**. They are slightly more general than pretopoi with W-sets as investigated in Moerdijk and Palmgren (2000), in that we do not require (exact) colimits. The correspondence between set-theoretic assumptions and categorical structures are investigated in more detail in a number of places, e.g. see Hofmann (1994, 1997); Abbott (2003).

We write **Set** for the chosen category of sets, and use $\Rightarrow$ for the functor category. I.e. while Set $\to$ Set is the type of operators on sets, **Set** $\Rightarrow$ **Set** is the functor category.

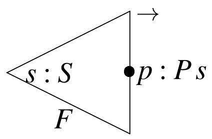
Given $A, B : \mathrm{Set}$ we write $A \cong B$ for the set of isomorphisms. Given $f : A \cong B$ we implicitly project f to $A \to B$ and use $f^{-1} : B \to A$ for the second component of the isomorphism. We use the fact that any pattern matching program which establishes a correspondence between a partition of its domain and a partition of its codomain is an isomorphism. E.g. to construct $f : A \times (B + C) \cong (A \times B) + (A \times C)$ we write

$$
\begin{aligned}
&f\,(a, \mathsf{inl}\, b) \rightleftharpoons \mathsf{inl}\,(a, b)\\
&f\,(a, \mathsf{inr}\, c) \rightleftharpoons \mathsf{inr}\,(a, c) \quad .
\end{aligned}
$$

Finally, in our presentation of proofs, we distinguish between the use of equations $=$ which hold definitionally and isomorphisms $\cong$ which we have proven. We will often mark an equational development with a directed $[\text{HINT}\rangle$ indicating the properties we exploit.

3 Containers Revisited

A container $F = (s : S \rhd P\, s)$ is given by a set of shapes $S : \mathrm{Set}$ and a family of positions $P : S \to \mathrm{Set}$. We illustrate such a container by drawing a *triangle diagram*, showing a typical shape s in the apex—the base represents the position set $P\, s$, where we show a typical position p, as follows:



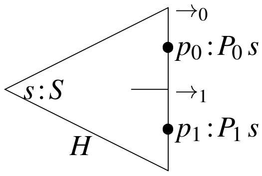
The extension of a container $[\![F]\!]$ is an operator on sets, given by

$$
\begin{aligned}
[\![F]\!] \;&:\; \mathrm{Set} \to \mathrm{Set}\\
[\![F]\!]\, X \;&:=\; (s : S;\; P\, s \to X) \quad .
\end{aligned}
$$

An element of $[\![F]\!]\, X$ thus consists of a choice $s : S$ of shape and a function f, attaching ‘payload’ from X to the positions in $P\, s$.

As an example consider the case of lists, $\mathsf{List} = (n : \mathsf{Nat} \rhd \mathsf{Fin}\, n)$, where $\mathsf{Fin}\, n = \{i < n\}$. For any $X : \mathrm{Set}$ an element $(n, f) : [\![\mathsf{List}]\!]\, X$ is given by $n : \mathsf{Nat}$ and a function $f : \mathsf{Fin}\, n \to X$ giving access to the n elements of the list. Here we regard the value n as giving the *shape* of the list.

In general we are interested in $n+1$-ary containers $(s : S \rhd P_0\, s, P_2\, s, \dots P_n\, s)$ where $P_i : S \to \mathrm{Set}$. To reduce clutter, we will restrict our attention to 2-ary containers $H = (s : S \rhd P_0\, s, P_1\, s)$ which we depict as follows:



The labels $\to_0$ and $\to_1$ give the correspondence between the position sets and the container's parameters which is trivial in the diagrams for 1-ary containers.

The extension of a 2-ary container is given by

$$\begin{aligned} [\![H]\!] &: \mathrm{Set} \to \mathrm{Set} \to \mathrm{Set} \\ [\![H]\!]\, X\, Y &= (s : S;\ P_1\, s \to X;\ P_2\, s \to Y) \ . \end{aligned}$$

We define a number of operations on containers: given a $C : \mathrm{Set}$ we define the constant container $\mathsf{K}\, C := (C \rhd 0)$ identity container $\mathsf{I} := (1 \rhd 1)$. Their extensions have the behaviour we expect

$$\begin{aligned} [\![\mathsf{K}\, C]\!]\, X &\cong C \\ [\![\mathsf{I}]\!]\, X &\cong X \ . \end{aligned}$$

Given $C : \mathrm{Set}$ and containers $F = (s : S \rhd P s)$, $G = (t : T \rhd Q\, t)$ we define

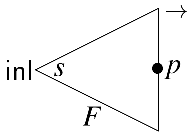
$$\begin{aligned} C \to F &= (f : C \to S \rhd c : C;\ P\,(f\, c)) \\ F + G &= \begin{pmatrix} \ \ \mathsf{inl}\, s : S \ \rhd P\, s \\ \mid \mathsf{inr}\, t : T \rhd Q\, t \end{pmatrix} \\ F \times G &= (s : S;\ t : T \ \rhd P\, s + Q\, t) \end{aligned}$$

and indeed, one may readily verify that their extensions behave correspondingly

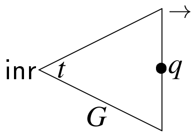
$$\begin{aligned} [\![C \to F]\!]\, X &\cong C \to [\![F]\!]\, X \\ [\![F + G]\!]\, X &\cong [\![F]\!]\, X + [\![G]\!]\, X \\ [\![F \times G]\!]\, X &\cong [\![F]\!]\, X \times [\![G]\!]\, X \ . \end{aligned}$$

We defer the definition of fixpoint operators until section 7.

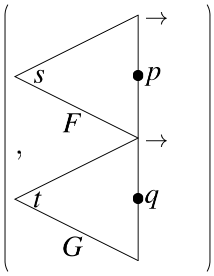
We can use triangle diagrams to illustrate the typical forms which container operators yield. For $F + G$, we have two forms



or



whilst for products we have one form with two parts



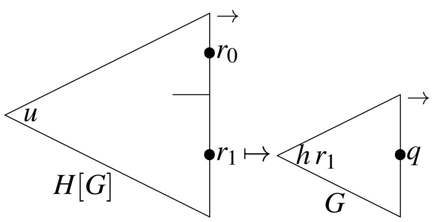
For the 2-ary case we introduce $\mathsf{P}_0 := (1 \triangleright 1, 0)$ and $\mathsf{P}_1 := (1 \triangleright 0, 1)$. We can weaken a 1-ary container $F = (s : S \triangleright P\,s)$ to a 2-ary container $\uparrow F := (s : S \triangleright P\,s, 0)$. We can compose containers: given a 2-ary container $H = (u{:}U \triangleright R_0\,u, R_1\,u)$ and a container $G = (t{:}T \triangleright Q\,t)$ we define

$$H[G] := \big(U;\ f : R_2\,u \to T \ \triangleright R_1\,u + (r_2 : R_2\,u;\ Q\,(f\,r_2))\big)$$

and indeed

$$\begin{aligned}
[\![\mathsf{P}_0]\!]\,X\,Y &\cong X \\
[\![\mathsf{P}_1]\!]\,X\,Y &\cong Y \\
[\![\uparrow F]\!]\,X\,Y &\cong [\![F]\!]\,X \\
[\![H[G]]\!]\,X &\cong [\![H]\!]\,X\,([\![G]\!]\,X) \ .
\end{aligned}$$

The typical form of $H[G]$ is shown thus:



Observe that payload can typically be found either at a ‘top’ position $r_0 : R_0\,u$, or at some point q inside a G attached ‘below’ r_1.

It is straightforward to generalise these combinators to n-ary containers: we introduce a family of projection operators P^n_i for $i < n$ (actually $\mathsf{I} = \mathsf{P}^1_0$), a family of weakening operators $\uparrow^i F$ weakening a n-ary container to $n+1$-ary, all the other operations can be lifted to n-ary containers. The composition operators $H[i = G]$ for an $(n+1)$-ary container H and an n-ary container G can also be viewed as a local definition or ‘let’ in a de Bruijn representation, as used in McBride (2001). In general we have all the ingredients to construct all strictly positive operators made from $+, \times, \to$ and constants. After giving the interpretation of fixpoint operators in section 7 this extends to arbitrarily nested strictly positive inductive and coinductive types. We will use type expressions with variables in strictly positive positions to construct containers, leaving the obvious translation to the de Bruijn operators discussed above implicit.

Containers form a category **Con**: given containers $F = (s:S \rhd Ps)$, $G = (t:T \rhd Q\,t)$ we define the homset $\mathbf{Con}(F, G) : \mathrm{Set}$ thus:

$$\mathbf{Con}(F, G) := (\sigma : S \to T;\ (s:S) \to Q\,(\sigma\, s) \to (P\,s)) \ .$$

Identity and composition are straightforward:

$$\begin{aligned} \mathrm{id}_F &:= (\lambda s.\, s,\, \lambda s\, p.\, p) \\ (\sigma', \psi') \cdot (\sigma, \psi) &:= (\lambda s.\, \sigma'(\sigma s), \lambda s\, r.\, \psi' s(\psi(\sigma s) r)) \ . \end{aligned}$$

This construction extends in a straightforward way to n-ary containers forming categories $\mathbf{Con}_n$ (where clearly $\mathbf{Con} = \mathbf{Con}_1$); substitution can then be regarded as a functor $-[-] : \mathbf{Con}_2 \Rightarrow \mathbf{Con} \Rightarrow \mathbf{Con}$.

We observe that $[\![-]\!]$ extends to morphisms, mapping container morphisms to natural transformations, giving rise to a functor $[\![-]\!] : \mathbf{Con} \Rightarrow \mathbf{Set} \Rightarrow \mathbf{Set}$ by

$$[\![(\sigma, \psi)]\!]\, X := \lambda(s, f).\, (\sigma\, s,\, \lambda q.\, f\,(\psi\, s\, q)) \ .$$

Indeed, every natural transformation between the extension of containers is uniquely determined by a container morphism.

Theorem 3.1 (Abbott et al., 2003a, theorem 3.4). *The extension functor* $[\![-]\!]$ *is full and faithful.* □

As a consequence we are able to observe

Corollary 3.2. Con *is bicartesian, i.e. has all finite products, coproducts and is distributive, and this structure is preserved by* $[\![-]\!]$. □

While $[\![-]\!]$ is full on morphisms, there are functors which are not in its image. A counterexample is $F\,X = (X \to 0) \to 0$: if there were a container $(s:S \rhd Ps)$ with extension $[\![(s:S \rhd Ps)]\!] \cong F$ then we can calculate $S = F\,1 \cong 1$. We have that $F\,2 \cong 1$ and hence $P \simeq 0$, however, $F\,0 = 0 \not\cong 0 \to 0$.

As a consequence of theorem 3.1 the extensions of two containers are naturally isomorphic iff they are isomorphic in **Con**. We use the following notation to define both components of a container isomorphism simultaneously. E.g. to show that $(G+H) \times F \cong (F \times G) + (F \times H)$ where $F = (s:S \rhd Ps)$, $G = (t:T \rhd Q\,t)$, $H = (u:U \rhd R\,u)$ we first expand the definitions for both sides:

$$(G+H) \times F = \begin{pmatrix} \mathsf{inl}\, t:T\ ;\ s:S \rhd Q\,t + P\,s \\ \mid \mathsf{inr}\, u:U;\ s:S \rhd R\,u + P\,s \end{pmatrix}$$

$$(F \times G) + (F \times H) = \begin{pmatrix} \mathsf{inl}\,(s:S;\ t:T) \rhd P\,s + Q\,t \\ \mid \mathsf{inr}\,(s:S, u:U) \rhd P\,s + R\,u \end{pmatrix}$$

we define the isomorphism $(\sigma, \psi) : (G+H) \times F \cong (F \times G) + (F \times H)$ by

$$
\begin{array}{ll}
\sigma\,(\mathsf{inl}\,t, s) & \rightleftharpoons \mathsf{inl}\,(s,t) \\
\quad \triangleright \left\{ \begin{array}{l} \mathsf{inr}\,p \\ \mathsf{inl}\,q \end{array} \right. \rightleftharpoons & \quad \triangleright \left\{ \begin{array}{l} \psi(\mathsf{inl}\,p) \\ \psi\,(\mathsf{inr}\,q) \end{array} \right. \\
\sigma\,(\mathsf{inr}\,u, s) & \rightleftharpoons \mathsf{inr}\,(s,u) \\
\quad \triangleright \left\{ \begin{array}{l} \mathsf{inr}\,p \\ \mathsf{inl}\,u \end{array} \right. \rightleftharpoons & \quad \triangleright \left\{ \begin{array}{l} \psi(\mathsf{inl}\,p) \\ \psi(\mathsf{inr}\,u) \end{array} \right.
\end{array}
$$

Note that in this example the movement of F from the right of $G+H$ to left of G and H is reflected in the morphisms of positions shown above.

4 Decidability

Our presentation of container structures makes it easy to refer to the positions within a given element. If $(s, f) : [\![(s : S \ \triangleright\ P\,s)]\!]\,X$, then $f\,p$ is the piece of payload at position $p : P\,s$. In order to explain the notion of 'one-hole context at p', we shall need to define the set of 'positions apart from p'.

Definition 4.1 (Complement). *For any $x{:}X$, we define the* complement *of x in X, written $X - x$, and the weakening map $|\cdot|_x : (X - x) \to X$ as follows:*

$$
\begin{array}{r}
X - x := (y{:}X;\ \mathsf{Eq}\,x\,y \to 0) \\
|(y, \_)|_x := y \quad .
\end{array}
$$

Where the set to which x belongs is clear, we may write $-x$ for $X - x$.

Moreover, a notion of one-hole context makes little sense unless it is equipped with linear substitution—'plugging something into the hole'—which necessitates the capacity to decide if some position p' is the hole or not. We shall therefore be particularly interested in position sets which admit equality testing.

Definition 4.2 (Decidability). *A set X is* decidable *if there exists*

$$
\mathsf{eq}_X : (x, y{:}X) \to \mathsf{Eq}\,x\,y + (\mathsf{Eq}\,x\,y \to 0) \quad .
$$

Note that any such eq_X must be unique: for any values x, y, both $\mathsf{Eq}\,x\,y$ and $\mathsf{Eq}\,x\,y \to 0$ have at most one inhabitant, and they cannot be inhabited simultaneously. Hence for any decidable X with $x, y{:}X$, we have $1 \cong \mathsf{Eq}\,x\,y + (\mathsf{Eq}\,x\,y \to 0)$ for any $x, y{:}X$. In particular, this enables us to partition a decidable set between any given element x and its complement.

We may now characterise the containers whose position sets are all decidable.

Definition 4.3 (Decidable container). *$(s\!:\!S \rhd P_1\, s, \ldots, P_n\, s)$ is a* decidable container *if for all $s\!:\!S$, each $P_i\, s$ is decidable.*

Note that the *shape* set of a decidable container need not be decidable, only the positions for each shape.

Proposition 4.4 (Complement Partition). *If X is decidable and $x\!:\!X$, then there exists $\Delta_x : X \cong (X - x) + 1$, such that $\Delta_x\, x = \mathsf{inr}\,()$*

Proof.

$$\begin{aligned}\Delta_x\, y := \;& \text{case } \mathsf{eq}_X\, x\, y \\ & \text{of } \mathsf{inl}\,(\mathsf{refl}\, x) \quad\quad\;\; \mapsto \mathsf{inr}\,() \\ & \quad\;\; \mathsf{inr}\,(p\!:\!\mathsf{Eq}\;\; x\, y \to 0) \mapsto \mathsf{inl}\,(y, p)\end{aligned}$$

The inverse is readily seen to exist. □

Hence given $x\!:\!X$ the patterns x and $|x'\!:\!X - x|_x$ partition a decidable X. We exploit these patterns in our presentation of programs. For example, Δ_x behaves as if defined

$$\begin{aligned}\Delta_x\, x &:= \mathsf{inr}\,() \\ \Delta_x\, \left|x'\right|_x &:= \mathsf{inl}\, x'\end{aligned}$$

We shall be making use of another example—the combinator $[t, f']$ which grafts $x \mapsto t$ onto some function f' defined over $X - x$.

Definition 4.5 (Grafting). *Given $x\!:\!X$ with X decidable, $T : X \to \mathrm{Set}$, $t : T\, x$ and $f' : (x'\!:\!X - x) \to T\; |x'|_x$ define $[t, f'] : (x\!:\!X) \to T\, x$ such that*

$$\begin{aligned}[t, f']\, x &:= t \\ [t, f']\, \left|x'\right|_x &:= f'\, x' \quad .\end{aligned}$$

Moreover, every $f : (y\!:\!X) \to T\, y$ is equal to $[f\, x,\, f \cdot |-|_x]$, so any function over X matches the pattern

$$[t : T\, x,\, f' : (x'\!:\!X - x) \to T\; \left|x'\right|_x] \quad .$$

This will prove useful when we need to analyse the functional components of shapes generated by the combinator $H[G]$.

The following lemma is an application of grafting and shows how decidability transfers along isomorphisms.

Lemma 4.6. *If X is decidable then for every Y there is an isomorphism of isomorphisms:*

$$(x\!:\!X;\; (X - x \cong Y)) \cong (X \cong Y + 1) \quad .$$

Proof. Working from left to right, given $(x : X, \Psi : X - x \cong P)$, construct $[\mathsf{inr}\,(), \mathsf{inl} \cdot \Psi]$ via the grafting function above, which has an inverse because $\mathsf{inr}\,()$ and $\mathsf{inl}\,(\Psi\, x')$ partition $Y + 1$.

To reverse this construction, given $\Phi : X \cong Y + 1$, we take $x = \Phi^{-1}\,(\mathsf{inr}\,())$ and construct $(x, \Psi : (X - x) \cong Y)$ where

$$\begin{array}{lll} \Psi\, x' := \text{case } \Phi\ |x'|_x & & \\ \quad\quad \text{of } \mathsf{inl}\, y & \mapsto y & \\ \quad\quad\quad \mathsf{inr}\,() & \text{impossible, because } \Phi\, x = \mathsf{inr}\,() & \end{array}$$

and, inverting,

$$\begin{array}{lll} \Psi^{-1}\, y := \text{case } \Phi^{-1}\,(\mathsf{inl}\, y) & & \\ \quad\quad \text{of } |x'|_x & \mapsto x' & \\ \quad\quad\quad x & \text{impossible, because } \Phi^{-1}\,(\mathsf{inr}\,()) = x. & \end{array}$$

As $\Phi = [\mathsf{inr}\,(), \mathsf{inl} \cdot \Psi]$, our two constructions are mutually inverse. $\square$

4.1 Closure of decidability and compound complements

Sets with at most one inhabitant, such as 0, 1, $\mathsf{Eq}\, x\, y$ and $\mathsf{Eq}\, x\, y \to 0$ are trivially decidable, with an empty complement.

We may establish the basic equality properties of coproducts as follows:

Proposition 4.7 (Coproduct preserves decidability). *If A and B are decidable, so is $A + B$.*

Proof. In each of the four possible cases, the task of deciding an equality on $A + B$ reduces either to a problem with a known solution, as we have:

$$\begin{aligned} \mathsf{Eq}\,(\mathsf{inl}\, a)\,(\mathsf{inl}\, a') &\cong \mathsf{Eq}\, a\, a' \\ \mathsf{Eq}\,(\mathsf{inl}\, a)\,(\mathsf{inr}\, b) &\cong 0 \\ \mathsf{Eq}\,(\mathsf{inr}\, b)\,(\mathsf{inl}\, a) &\cong 0 \\ \mathsf{Eq}\,(\mathsf{inr}\, b)\,(\mathsf{inr}\, b') &\cong \mathsf{Eq}\, b\, b' \quad . \end{aligned}$$

$\square$

Proposition 4.8 (Σ preserves decidability). *If A is decidable, and $C\, a$ is decidable for all $a : A$, then $(a : A;\ C\, a)$ is decidable. In particular $A - a$ is decidable.*

Proof. As above, an equality decision on pairs reduces to a pair of equality decisions, given that

$$\mathsf{Eq}\,(a_1, c_1)\,(a_2, c_2) \cong (\mathsf{Eq}\, a_1\, a_2;\ \mathsf{Eq}\, c_1\, c_2) \quad .$$

$\square$

We shall shortly present our construction of one-hole contexts via complement sets. In the proofs which follow, we shall need to exploit the following isomorphisms, which show us how we can simplify complements, given some information about the element they exclude. We indicate the use of these results by the hint $[\text{SIMPLIFY } -\rangle$.

Proposition 4.9 (Complement simplification). *If D is decidable, we have:*

$$\begin{aligned}
0 - x &\cong 0 \\
(A+B) - (\mathsf{inl}\, a) &\cong (A-a)+B \\
(A+B) - (\mathsf{inr}\, b) &\cong A+(B-b) \\
1 - () &\cong 0 \\
(x{:}D;\ C\,x) - (d,c) &\cong (d' : (D-d);\ C\,\left|d'\right|_d) + (C\,d - c) \\
(\mathsf{Eq}\,x\,y) - p &\cong 0 \\
(\mathsf{Eq}\,x\,y \to 0) - p &\cong 0 \ .
\end{aligned}$$

Proof. Sets with at most one inhabitant have 0 as complement. This leaves the laws for coproducts and Σ-sets with decidable first component. For inl:

$$\begin{aligned}
&(A+B) - (\mathsf{inl}\, a) \\
&\qquad [\text{EXPAND } -\rangle \\
=\ & (x{:}A+B;\ \mathsf{Eq}\,x\,(\mathsf{inl}\,a) \to 0) \\
&\qquad [\text{DISTRIBUTE } \Sigma,+\rangle \\
\cong\ & (a'{:}A;\ \mathsf{Eq}\,(\mathsf{inl}\,a')\,(\mathsf{inl}\,a) \to 0) + (b{:}B;\ \mathsf{Eq}\,(\mathsf{inr}\,b)\,(\mathsf{inl}\,a) \to 0) \\
&\qquad [\text{SIMPLIFY EQUATIONS}\rangle \\
\cong\ & (a'{:}A;\ \mathsf{Eq}\,a\,a' \to 0) + (b{:}B;\ 0 \to 0) \\
&\qquad [\text{DEFINITION OF } -,\ \text{ALGEBRA}\rangle \\
\cong\ & (A-a)+B
\end{aligned}$$

The proof for inr is similar. For Σ-sets:

$$\begin{aligned}
&(x{:}D;\ C\,x) - (d,c) \\
&\qquad [\text{EXPAND } -\rangle \\
=\ & ((x{:}D;\ y{:}C\,x);\ \mathsf{Eq}\,(x,y)\,(d,c) \to 0) \\
&\qquad [\text{SIMPLIFY EQUATION}\rangle \\
\cong\ & (x;\ y;\ (\mathsf{Eq}\,x\,d;\ \mathsf{Eq}\,y\,c) \to 0) \\
&\qquad [\text{DECIDE IF } x = d\rangle \\
=\ & \begin{array}{l} |d' : (D-d)|_d;\ y{:}C\,|d'|_d;\ (\mathsf{Eq}\,|d'|_d\,d;\ \mathsf{Eq}\,y\,c) \to 0 \\ \mid d \qquad\qquad\quad ;\ y{:}C\,d \quad\ ;\ (\mathsf{Eq}\,d\,d;\ \mathsf{Eq}\,y\,c) \to 0 \end{array} \\
&\qquad [\text{SIMPLIFY EQUATIONS}\rangle \\
\cong\ & \begin{array}{l} |d'|_d;\ y;\ 0 \to 0 \\ \mid d \quad ;\ y;\ \mathsf{Eq}\,c\,y \to 0 \end{array} \\
&\qquad [\text{ALGEBRA}\rangle \\
\cong\ & (d' : (D-d);\ C\,|d'|_d) + (C\,d - c) \ .
\end{aligned}$$

□

Returning to our original motivation, we now know enough to establish that the relevant standard container operators preserve decidability of containers.

Theorem 4.10 (Container decidability closure). *Decidability of containers is closed under* K, I, P, $+$, $\times$ *and* $-[-]$.

Proof. Inspecting the position families generated by these operators, they use only set-forming operations which preserve decidability, as we have established above. $\square$

5 Derivatives Universally

In analogy with the classical development of derivatives, we will first specify derivatives by a universal property and then present an implementation for decidable containers which satisfies our specification. In our context the notion corresponding to linear functions are cartesian morphisms which then can be used to specify derivatives.

A cartesian morphism is a container morphism whose action on positions is a family of isomorphisms, i.e. we define the category $\mathbf{Con}^{\multimap}$ which has the same objects as $\mathbf{Con}$ but with homsets $\mathbf{Con}^{\multimap}(F, G) : \mathrm{Set}$ given by

$$\mathbf{Con}^{\multimap}(F, G) := (\sigma : S \to T;\ (s : S) \to Q\,(\sigma\, s) \cong (P\, s))$$

for $F = (s : S \rhd P s)$, $G = (t : T \rhd Q\, t)$. The identity and composition are inherited from $\mathbf{Con}$.

We can now specify the derivative of a container as the linear approximations of its tangents, so that the derivative of G is defined to be equipped with an isomorphism

$$\mathbf{Con}^{\multimap}(F \times \mathsf{I}, G) \cong \mathbf{Con}^{\multimap}(F, \partial G)$$

natural in F. Naturality in F means that given cartesian container morphisms $\alpha : \mathbf{Con}^{\multimap}(H, F)$, $\beta : \mathbf{Con}^{\multimap}(F \times \mathsf{I}, G)$ and writing θ_F for the isomorphism above, there is an equality $\theta_H(\beta \cdot (\alpha \times \mathsf{I})) = (\theta_F \beta) \cdot \alpha$.

More concisely, we can say that the derivative of G is a cartesian container morphism $s^G : \mathbf{Con}^{\multimap}(\partial G \times \mathsf{I}, G)$ which is a *universal arrow* from the functor $- \times \mathsf{I} : \mathbf{Con}^{\multimap} \Rightarrow \mathbf{Con}^{\multimap}$ to G.

We can explicitely calculate the derivative of a decidable container:

$$\partial(s : S \rhd P\, s) := (s : S;\ p : P\, s \rhd P\, s - p)\ .$$

For n-ary containers the specification is

$$\mathbf{Con}^{\multimap}(F \times P_i, G) \cong \mathbf{Con}^{\multimap}(F, \partial_i G)$$

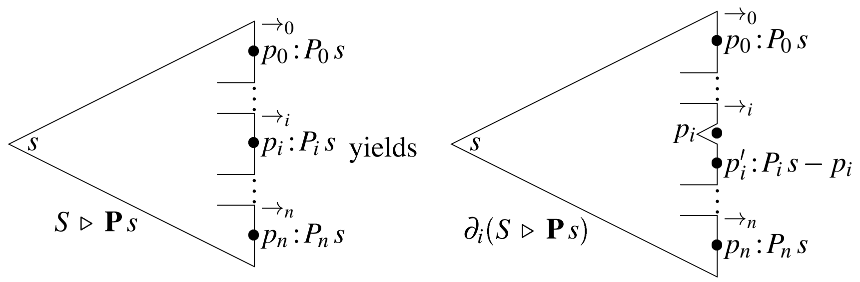
natural in F and the construction is given by the following:

Definition 5.1 (Derivative). *We define the operator ∂_i on a decidable $n+1$-ary containers as follows:*

$$\begin{aligned}&\partial_i(s:S \triangleright P_0\,s, \ldots, P_n\,s)\\ &:= (s:S;\; p_i:P_i\,s \triangleright P_0\,s, \ldots, P_{i-1}\,s, P_i\,s - p_i, P_{i+1}\,s, \ldots, P_n\,s)\ .\end{aligned}$$

That is, the shape of a derivative includes the choice of position for the hole; the hole is then excluded from the derivative's positions.

We illustrate this by a triangle diagram with a position cut out, like this:



Let us now verify that our implementation of derivatives satisfies the specification. To avoid unnecessary clutter we only consider the case of the unary derivative — the generalisation to the n-ary case is straightforward.

Theorem 5.2. *If G is a decidable container then there is an isomorphism*

$$\mathbf{Con}^{\multimap}(F \times \mathsf{I}, G) \cong \mathbf{Con}^{\multimap}(F, \partial G)$$

natural in F.

Proof. Let $G = (t:T \triangleright Q\,t)$, $F = (s:S \triangleright P s)$ and expand the left hand side

$$\begin{aligned}&\mathbf{Con}^{\multimap}(F \times \mathsf{I}, G)\\ &\qquad [\text{EXPAND } F, G\rangle\\ &= \mathbf{Con}^{\multimap}((s:S \triangleright P s) \times (1 \triangleright 1), (t:T \triangleright Q\,t))\\ &\qquad [\text{EXPAND } \times,\ \Sigma \text{ ABSORBS } 1\rangle\\ &\cong \mathbf{Con}^{\multimap}((s:S \triangleright P\,s + 1), (t:T \triangleright Q\,t))\\ &\qquad \left[\text{EXPAND } \mathbf{Con}^{\multimap}\right\rangle\\ &= (\sigma:S \to T;\ (s:S) \to (Q(\sigma\,s) \cong P\,s + 1))\\ &\qquad [\text{FACTOR OUT } S\rangle\\ &\cong (s:S) \to (t:T;\ Q\,t \cong P\,s + 1)\end{aligned}$$

and the right hand side

$$
\begin{aligned}
&\mathbf{Con}^{\multimap}(F, \partial G) \\
&\qquad [\text{EXPAND } F,G,\partial\rangle \\
&= \mathbf{Con}^{\multimap}((s:S \triangleright Ps), (t:T;\ q:Q\,t \triangleright Q\,t - q)) \\
&\qquad \left[\text{EXPAND } \mathbf{Con}^{\multimap}, \text{ SPLIT } \tau\right\rangle \\
&= (\lambda s.\, (\tau_t\, s, \tau_q\, s) : S \to (t:T;\ Qt);\ (s:S) \to (Q\,(\tau_t\, s) - \tau_q\, s \cong P\,s)) \\
&\qquad [\text{FACTOR OUT } S\rangle \\
&\cong (s:S) \to (t:T;\ q:Q\,t;\ Q\,t - q \cong P\,s) \ \ .
\end{aligned}
$$

Now by appealing to lemma 4.6 the isomorphism is established.

To show that this is a natural bijection, it is instructive to chase the fate of $\mathrm{id}_{\partial G}$ through the bijection above to a morphism $s^F : \mathbf{Con}^{\multimap}(\partial G \times \mathsf{I}, G)$. First specialise the expansion of the right hand type to

$$
\mathbf{Con}^{\multimap}(\partial G \times \mathsf{I}, G) \cong (t_1 : T;\ q : Q\,t_1) \to (t_2 : T;\ Q\,t_2 \cong (Q\,t_1 - q) + 1)
$$

and then we can compute $s^F = \lambda t q.\, (t, \Delta_q)$ on the right hand side of this isomorphism. Naturality then follows by observing that the isomorphism $\mathbf{Con}^{\multimap}(F, \partial G) \to \mathbf{Con}^{\multimap}(F \times \mathsf{I}, G)$ is induced by composition with s^F, taking $\alpha : \mathbf{Con}^{\multimap}(F, \partial G)$ to $s^F \cdot (\alpha \times \mathsf{I})$. □

6 Laws of Derivatives

Let us now establish that ∂ satisfies the laws we expect. The arguments we give here extend readily to n-ary containers. The proofs we present all follow the same plan:

1. expand the definitions of the containers, container operations and ∂ on both sides of the law to be proven;
2. simplify the instances of $P - p$ which arise from ∂ in each case by the rules established in Proposition 4.9;
3. ensure that the shapes on each side have been analysed into cases carrying the same information, and that for each case, the possible positions also correspond.

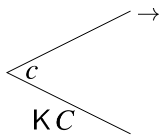
As we develop each side, we indicate what we are expanding or how we are simplifying with a directed [HINT⟩. To reduce clutter, we shall often omit repeated type annotations on variables where they are unchanged from their previous explicit usage. This may be unusual mathematical practice, but it is a commonplace of programming. Its advantage here over the point-free style is that it keeps explicit the structural dependencies which are crucial to this work, whilst leaving implicit only what has already been stated. Similarly, we shall often write $(A - a)$ as just $(-a)$, if we have already introduced $a\!:\!A$. Hence, for example, we might write

$$\begin{aligned}
&\partial(s\!:\!S \triangleright Ps)\\
&\qquad[\text{EXPAND } \partial\rangle\\
&= (s;\ p\!:\!Ps \triangleright -p)\ \ .
\end{aligned}$$

When we have developed both sides, we shall illustrate the intuition behind the expanded forms by drawing triangle diagrams for the possible forms of the structure being differentiated, and the derivatives they yield. The latter diagrams indicate the shape and position information for each possible configuration with the same identifiers which appear in the proof. One may then readily check that the two sides correspond both to the intuition given by the diagrams and to each other.

Proposition 6.1 (Constant Rule). $\partial(\mathsf{K}\,C) \cong \mathsf{K}\,0$.

Proof. Constant containers have shapes but no positions for payload.



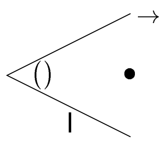
Hence we should expect an empty derivative. Developing both sides:

$$\begin{array}{l|l}
\partial(\mathsf{K}\,C) & \mathsf{K}\,0\\
\qquad[\text{EXPAND } \mathsf{K}\rangle & \qquad[\text{EXPAND } \mathsf{K}\rangle\\
=\ \partial(c\!:\!C \triangleright 0) & =\ (0 \triangleright 0)\\
\qquad[\text{EXPAND } \partial\rangle & \\
=\ (c;\ x\!:\!0 \triangleright -x) & \\
\qquad[\text{SIMPLIFY } -\rangle & \\
\cong\ (c;\ x\!:\!0 \triangleright 0) & \\
\qquad[\text{DISTRIBUTE } \Sigma, 0\rangle & \\
\cong\ (0 \triangleright 0) &
\end{array}$$

□

Proposition 6.2 (Identity Rule). $\partial\mathsf{I} \cong \mathsf{K}\,1$.

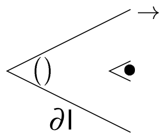
Proof. The identity container has one shape and one position.



Developing both sides:

$\begin{aligned} & \partial \mathsf{I} \\ & \quad [\text{EXPAND } \mathsf{I}\rangle \\ = \ & \partial(1 \triangleright 1) \\ & \quad [\text{EXPAND } \partial\rangle \\ = \ & (1;\ () \triangleright -()) \\ & \quad [\text{SIMPLIFY } -\rangle \\ \cong \ & (1;\ () \triangleright 0) \\ \cong \ & (1 \triangleright 0) \end{aligned}$	$\begin{aligned} & \mathsf{K}\,1 \\ & \quad [\text{EXPAND } \mathsf{I}\rangle \\ = \ & (1 \triangleright 0) \end{aligned}$

The derivative is unique:

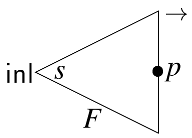


□

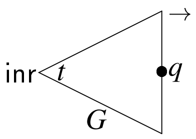
In the next three proofs, we take F and G to be typical 1-ary containers $(s\!:\!S \triangleright Ps)$ and $(t\!:\!T \triangleright Q\,t)$ respectively, whilst H is a typical 2-ary container, $(u\!:\!U \ \triangleright R_0\,u, R_1\,u)$.

Proposition 6.3 (Sum Rule). $\partial(F+G) \cong \partial F + \partial G$.

Proof. $F+G$ yields two typical forms, morally



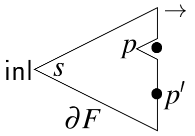
or



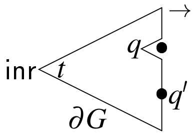
Developing the left-hand side:

$$\begin{aligned} & \partial(F+G) \\ & \quad [\text{EXPAND } F, G\rangle \\ = \ & \partial((s\!:\!S \triangleright Ps) + (t\!:\!T \triangleright Q\,t)) \\ & \quad [\text{EXPAND } +\rangle \\ = \ & \partial \begin{pmatrix} \ \ \mathsf{inl}\,s \triangleright P\,s \\ \mid \mathsf{inr}\,t \triangleright Q\,t \end{pmatrix} \\ & \quad [\text{EXPAND } \partial\rangle \\ = \ & \begin{pmatrix} \ \ \mathsf{inl}\,s;\ p\!:\!P\,s \triangleright -p \\ \mid \mathsf{inr}\,t;\ q\!:\!Q\,t \triangleright -q \end{pmatrix} \\ & \quad [\text{DISTRIBUTE } \Sigma, +\rangle \\ \cong \ & \begin{pmatrix} \ \ \mathsf{inl}\,(s,p) \triangleright p'\!:\!-p \\ \mid \mathsf{inr}\,(t,q) \triangleright q'\!:\!-q \end{pmatrix} \end{aligned}$$

As expected, we have the following possibilities:



or



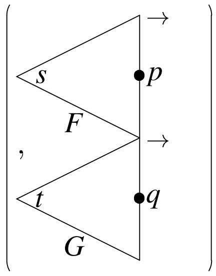
Developing the right-hand side:

$$
\begin{aligned}
& \partial F + \partial G \\
& \qquad [\text{EXPAND } F, G\rangle \\
=\ & \partial(s{:}S \triangleright Ps) + \partial(t{:}T \triangleright Q\,t) \\
& \qquad [\text{EXPAND } \partial\rangle \\
=\ & (s;\ p{:}P\,s \triangleright -p) + (t;\ q{:}Q\,t \triangleright -q) \\
& \qquad [\text{EXPAND } +\rangle \\
=\ & \begin{pmatrix} \ \ \mathsf{inl}\,(s,p) \triangleright p'{:}-p \\ \mid \mathsf{inr}\,(t,q) \triangleright q'{:}-q \end{pmatrix}
\end{aligned}
$$

We have arrived at the same analysis. □

Proposition 6.4 (Product Rule). $\partial(F \times G) \cong \partial F \times G + F \times \partial G.$

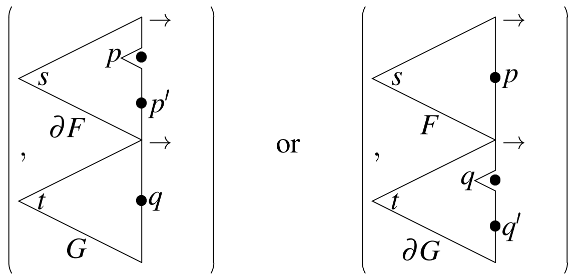
Proof. Products have one typical form, but with two typical kinds of position.



Developing the left-hand side:

$$
\begin{aligned}
& \partial(F \times G) \\
& \qquad [\text{EXPAND } F, G\rangle \\
=\ & \partial((s{:}S \triangleright Ps) \times (t{:}T \triangleright Q\,t)) \\
& \qquad [\text{EXPAND } \times\rangle \\
=\ & \partial(s;\ t \triangleright P\,s + Q\,t) \\
& \qquad [\text{EXPAND } \partial\rangle \\
=\ & \begin{pmatrix} \ \ s;\ t;\ \mathsf{inl}\,p{:}P\,s \triangleright (P\,s + Q\,t) - \mathsf{inl}\,p \\ \mid s;\ t;\ \mathsf{inr}\,q{:}Q\,t \triangleright (P\,s + Q\,t) - \mathsf{inr}\,q \end{pmatrix} \\
& \qquad [\text{SIMPLIFY } -\rangle \\
\cong\ & \begin{pmatrix} \ \ s;\ t;\ \mathsf{inl}\,p \triangleright p'{:}(P\,s - p) + Q\,t \\ \mid s;\ t;\ \mathsf{inr}\,q \triangleright P\,s + q'{:}(Q\,t - q) \end{pmatrix}
\end{aligned}
$$

This choice is illustrated as follows:



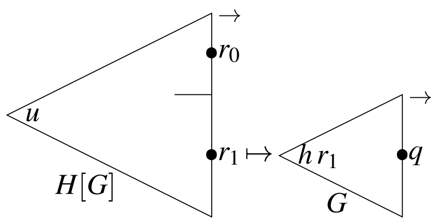
Developing the right-hand side:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\partial F \times G + F \times \partial G} \\
& [\text{EXPAND } F, G\rangle \\
= & \partial(s\!:\!S \triangleright Ps) \times (t\!:\!T \triangleright Q\,t) + (s\!:\!S \triangleright Ps) \times \partial(t\!:\!T \triangleright Q\,t) \\
& [\text{EXPAND } \partial\rangle \\
= & (s;\ p\!:\!P\,s \triangleright -p) \times (t \triangleright Q\,t) + (s \triangleright Ps) \times (t;\ q\!:\!Q\,t \triangleright -q) \\
& [\text{EXPAND } \times\rangle \\
= & (s;\ p;\ t \triangleright (-p) + Q\,t) + (s;\ t;\ q \triangleright P\,s + (-q)) \\
& [\text{EXPAND } +\rangle \\
= & \left(\begin{array}{l} \ \ \mathsf{inl}\,(s,p,t) \triangleright (-p) + Q\,t \\ \mid \mathsf{inr}\,(s,t,q) \triangleright P\,s + (-q) \end{array}\right) \\
& [\text{FACTORIZE}\rangle \\
\cong & \left(\begin{array}{l} \ \ s;\ t;\ \mathsf{inl}\,p \triangleright p' : (P\,s - p) + Q\,t \\ \mid s;\ t;\ \mathsf{inr}\,q \triangleright P\,s + q' : (Q\,t - q) \end{array}\right)
\end{array}
$$

as required. □

Proposition 6.5 (Chain Rule). $\partial(H[G]) \cong \partial_0 H[G] + \partial_1 H[G] \times \partial G.$

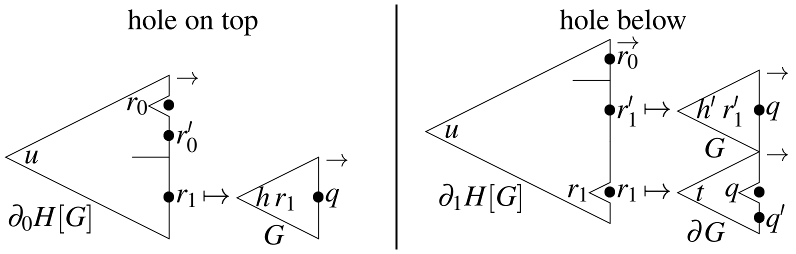
Proof. $H[G]$ containers capture this general form



with two typical positions for payload—either at the top level, corresponding to the first parameter of H, or within a G container located at a position corresponding to the second parameter of H. We shall see this analysis emerge as we develop the left-hand-side:

$$
\begin{array}{l}
\partial(H[G]) \\
\qquad [\textsc{expand } H, G\rangle \\
= \ \partial((u\!:\!U \ \triangleright \ R_0\,u, R_1\,u)[t\!:\!T \triangleright Q\,t]) \\
\qquad [\textsc{expand } -[-]\rangle \\
= \ \partial(u;\ h\!:\!R_1\,u \to T \ \triangleright \ R_0\,u + (r\!:\!R_1\,u;\ Q\,(h\,r))) \\
\qquad [\textsc{expand } \partial\rangle \\
= \left(\begin{array}{ll} \ u;\ h;\ \mathsf{inl}\,r_0\!:\!R_0\,u & \triangleright\ (R_0\,u + (r;\ Q\,(h\,r))) - \mathsf{inl}\,r_0 \\ \mid u;\ h;\ \mathsf{inr}\,(r_1\!:\!R_1\,u, q\!:\!Q\,(h\,r_1)) & \triangleright\ (R_0\,u + (r;\ Q\,(h\,r))) - \mathsf{inr}\,(r_1, q) \end{array}\right) \\
\qquad [\textsc{distribute } h \textsc{ inside } + \textsc{ in order to}\ldots\rangle \\
\cong \left(\begin{array}{ll} \ u;\ \mathsf{inl}\,(h, r_0) & \triangleright\ (R_0\,u + (r;\ Q\,(h\,r))) - \mathsf{inl}\,r_0 \\ \mid u;\ \mathsf{inr}\,(r_1, h, q) & \triangleright\ (R_0\,u + (r;\ Q\,(h\,r))) - \mathsf{inr}\,(r_1, q) \end{array}\right) \\
\qquad [\ldots \textsc{split } h \textsc{ at } r_1 \textsc{ on second line}\rangle \\
= \left(\begin{array}{l} \ u;\ \mathsf{inl}\,(h, r_0) \\ \qquad \triangleright\ (R_0\,u + (r;\ Q\,(h\,r))) - \mathsf{inl}\,r_0 \\ \mid u;\ \mathsf{inr}\,(r_1, [t\!:\!T, h'\!:\!(-r_1) \to T], q\!:\!Q\,t) \\ \qquad \triangleright\ (R_0\,u + (r;\ Q\,([t, h']\,r))) - \mathsf{inr}\,(r_1, q) \end{array}\right) \\
\qquad [\textsc{simplify } -\rangle \\
\cong \left(\begin{array}{ll} \ u;\ \mathsf{inl}\,(h, r_0) & \triangleright\ (r_0'\!:\!-r_0) + (r_1\!:\!R_1\,u;\ q\!:\!Q\,(h\,r_1)) \\ \mid u;\ \mathsf{inr}\,(r_1, [t, h'], q) & \triangleright\ (r_0\!:\!R_0\,u) + (r_1'\!:\!-r_1;\ q\!:\!Q\,(h'\,r_1')) + (q'\!:\!-q) \end{array}\right)
\end{array}
$$

The effect of the above is that, depending on which parameter of H accounts for the hole's position, the remaining positions can avoid it in a variety of ways, as shown below:



If the hole is on top (r_0 in the first case), a different position can be either a different r_0' on top or any q below any r_1. In the second case, the hole is at q below r_1, and may be avoided in three ways—choose r_0 on top, choose any q below a different r_1', or choose a different q' under r_1. This is the analysis which the right-hand side makes explicit:

$$
\begin{array}{rl}
& \partial_0 H[G] + \partial_1 H[G] \times \partial G \\
& \quad [\text{EXPAND } H, G\rangle \\
= & \quad \partial_0(u{:}U \rhd R_0\,u, R_1\,u)[t{:}T \rhd Q\,t] \\
& + \partial_1(u{:}U \rhd R_0\,u, R_1\,u)[t{:}T \rhd Q\,t] \times \partial(t{:}T \rhd Q\,t) \\
& \quad [\text{EXPAND } \partial_0, \partial_1, \partial\rangle \\
= & \quad (u;\ r_0{:}R_0\,u \rhd -r_0, R_1\,u)[t \rhd Q\,t] \\
& + (u;\ r_1{:}R_1\,u \rhd R_0\,u, -r_1)[t \rhd Q\,t] \times (t;\ q{:}Q\,t \rhd -q) \\
& \quad [\text{EXPAND } -[-]\rangle \\
& \quad (u;\ r_0;\ h{:}R_1\,u \to T \rhd (-r_0) + (r_1{:}R_1\,u;\ Q\,(h\,r_1))) \\
= & + \quad (u;\ r_1;\ h'{:}(-r_1) \to T \rhd R_0\,u + (r_1'{:}-r_1;\ Q\,(h'\,r_1'))) \\
& \quad \times\ (t;\ q \rhd -q) \\
& \quad [\text{EXPAND } \times\rangle \\
= & \quad (u;\ r_0;\ h \rhd (-r_0) + (r_1;\ Q\,(h\,r_1))) \\
& + (u;\ r_1;\ h';\ t;\ q \rhd R_0\,u + (r_1';\ Q\,(h'\,r_1')) + (-q)) \\
& \quad [\text{EXPAND } +\rangle \\
= & \left(\begin{array}{ll} \mathsf{inl}\,(u, r_0, h) & \rhd (r_0'{:}-r_0) + (r_1;\ q{:}Q\,(h\,r_1)) \\ \mid \mathsf{inr}\,(u, r_1, h', t, q) & \rhd (r_0{:}R_0\,u) + (r_1';\ q'{:}Q\,(h'\,r_1')) + (q'{:}-q) \end{array}\right)
\end{array}
$$

The two sides, so developed, are isomorphic by application of basic algebraic laws. □

7 Fixpoints of Containers

Given a 2-ary container $H = u{:}U \rhd R_0\,u, R_1\,u$ and ϕ is either μ or ν we define a container ϕH

$$\phi H = (w : \Phi \rhd \mathsf{Pos}_H w)$$

where $\Phi = \phi X.\,(u{:}U;(R_2\,u) \to X)$ is $\mathsf{W}\,U\,R_2$ ($\mathsf{M}\,U\,R_2$ resp.) with

$$\mathsf{sup} : (u{:}U) \to (R_2\,u \to \Phi) \to \Phi$$

and $\mathsf{Pos}_H : \Phi \to \mathrm{Set}$ is defined inductively by

$$\frac{r_1 : R_1\,u}{\mathsf{top}\,r_1 : \mathsf{Pos}_H\,(\mathsf{sup}\,u\,f)} \qquad \frac{r_2 : R_2\,u \quad x : \mathsf{Pos}_H\,(f\,r_2)}{\mathsf{below}\,r_2\,x : \mathsf{Pos}_H\,(\mathsf{sup}\,u\,f)}$$

We are able to derive an isomorphism $\mathsf{in}_H = (\sigma, \psi)$ such that in_H represents the constructor in the initial algebra case and in_H^{-1} the destructor in the terminal algebra case:

$$\mathsf{in}_H : H[\phi\,H] \cong_{\mathbf{Con}} \phi\,H$$

by

$$\begin{array}{ll} \sigma(u,f) \rhd \mathsf{inl}\, r_1 & \sup u\, f \rhd \psi(\mathsf{top}\, r_1) \\ \mid \sigma(u,f) \rhd \mathsf{inr}(r_2,x) & \mid \sup u\, f \rhd \psi(\mathsf{below}\, r_2\, x) \end{array} \rightleftharpoons$$

using that

$$H[\phi H] = (u\!:\!U; f\!:\!R_2\, s \to w\!:\!\Phi \rhd R_1\, u + (R_2\!:\!R_2\, u; \mathsf{Pos}_H\,(f\, r_2)))$$

In Abbott et al. (2004b) we show that the names μH and νH are justified, i.e. containers are closed under constructing initial algebras and terminal coalgebras of their extension

Theorem 7.1. *Given H as above we have:*

1. *$(\llbracket \mu H \rrbracket X, \llbracket \mathsf{in}_H \rrbracket X)$ is the initial $\llbracket H \rrbracket X$ algebra.*
2. *$(\llbracket \nu H \rrbracket X, \llbracket \mathsf{in}_H^{-1} \rrbracket X)$ is the terminal $\llbracket H \rrbracket X$ coalgebra.*

The assignment is natural in X.

Note that although the theorem holds in the initial and terminal case, it is not true for any fixpoint.

All the ingredients of this construction can be derived from W-types: In Abbott (2003); Abbott et al. (2004a) we have shown that the inductive family Pos_H is definable using W-types and in Abbott et al. (2004b) we show that M-types are derivable from W-types. In particular we do not assume the presence of a universe.

To calculate the derivative of fixpoints, we will need a lemma to characterise complements of Pos_H:

Lemma 7.2.

$$\begin{aligned} \mathsf{Pos}_H(\sup u\, f) - \mathsf{top}\, r_0 &\cong (R_0\, u - r) + (r_1\!:\!R_1\, u;\ \mathsf{Pos}_H\,(f\, r_1)) \\ \mathsf{Pos}_H(\sup u\, f) - \mathsf{below}\, r_1\, x &\cong (R_0\, u) + (r_1'\!:\!R_1\, u - r_1;\ \mathsf{Pos}_H(f\, \left|r_1'\right|_{r_1})) \\ &\quad + (\mathsf{Pos}_H(f\, r_2) - x) \end{aligned}$$

Proof. By expanding the fixpoint and then applying prop 4.9.

8 Derivatives of Fixpoints

Let ϕ be either μ or ν. Let us now differentiate the fixpoint container ϕH where $H = (u\!:\!U \rhd R_0\, u, R_1\, u)$.

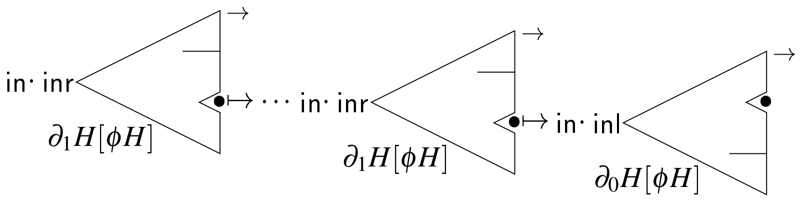
As above, let Φ be $\mathsf{W}\, U\, R_1$ or $\mathsf{M}\, U\, R_1$, corresponding to the choice of ϕ, and recall that

$$\phi H = (w\!:\!\Phi \rhd \mathsf{Pos}_H\, w)$$

Proposition 8.1 (Fixpoint rule). $\partial(\phi H) \cong \mu H'$ *where*

$$H' = (\uparrow \partial_0 H[\phi H]) + (\uparrow \partial_1 H[\phi H]) \times \mathsf{P}_1$$

In effect, an X's context is a finite sequence of steps recording the context of a sub-ϕH inside a ϕH, leading us to the node where the X was, and terminated by its context within that node. A typical element thus resembles the following:



We can sketch the argument by expanding fixpoints and applying the chain rule.

$$\begin{array}{ccl}
\partial(\phi H) & \cong \partial(H[\phi H]) \cong & \partial_1 H[\phi H] + \partial_2 H[\phi H] \times \ \partial(\phi H) \\
\cong & & \qquad\qquad\qquad\qquad\qquad\quad \cong \\
\mu H' & \cong & \partial_1 H[\phi H] + \partial_2 H[\phi H] \times \ \mu H'
\end{array}$$

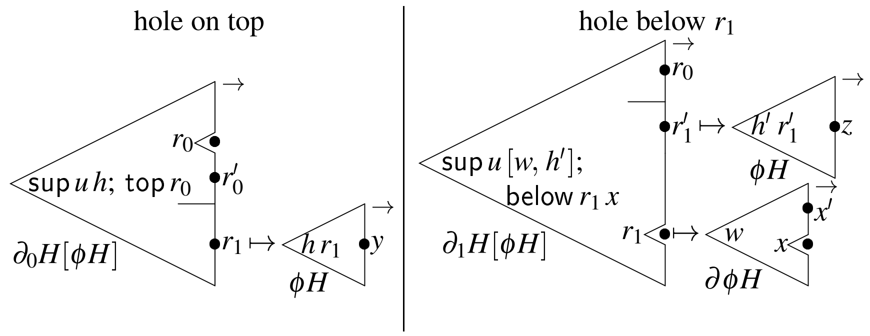
There is an obvious candidate for a recursive isomorphism between the two, but does it make sense? We should be optimistic, because the shapes in $\partial(\phi H)$ include positions from ϕH which are inductively defined regardless of whether ϕ is μ or ν, whilst $\mu H'$ is clearly inductive. We now make this intuition precise.

Proof. Following our usual procedure, we develop the left-hand side by expanding definitions and simplifying complements:

$$\begin{array}{ll}
\multicolumn{2}{l}{\partial(\phi H)} \\
& [\text{EXPAND } \phi\rangle \\
= & \partial(w\!:\!\Phi \triangleright \mathsf{Pos}_H\, w) \\
& [\text{EXPAND } \partial\rangle \\
= & w\!:\!\Phi;\ x\!:\!\mathsf{Pos}_H\, w \triangleright -x \\
& [\text{MATCH } x\rangle \\
= & \left(\begin{array}{l}
\mathsf{sup}\, u\, h;\ \mathsf{top}\, r_0\!:\!R_0\, u \triangleright \mathsf{Pos}_H\,(\mathsf{sup}\, u\, h) - (\mathsf{top}\, r_0) \\
\mid \mathsf{sup}\, u\, h;\ \mathsf{below}\, r_1\, x \ \ \triangleright \mathsf{Pos}_H\,(\mathsf{sup}\, u\, h) - (\mathsf{below}\, r_1\, x)
\end{array}\right) \\
& [\text{ON SECOND LINE, SPLIT } h \text{ AT } r_1\rangle \\
= & \left(\begin{array}{l}
\mathsf{sup}\, u\, h \qquad\qquad\qquad\qquad\ ;\ \mathsf{top}\, r_0 \qquad \triangleright \mathsf{Pos}_H\,(\mathsf{sup}\, u\, h) - (\mathsf{top}\, r_0) \\
\mid \mathsf{sup}\, u\,[w, h'\!:\!(-r_1) \to \Phi];\ \mathsf{below}\, r_1\, x \triangleright \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{Pos}_H\,(\mathsf{sup}\, u\,[w, h']) - (\mathsf{below}\, r_1\, x)
\end{array}\right) \\
& [\text{SIMPLIFY } -\rangle \\
\cong & \left(\begin{array}{l}
\mathsf{sup}\, u\, h \quad\ ;\ \mathsf{top}\, r_0 \quad\ \triangleright (r_0'\!:\!-r_0) + (r_1\!:\!R_1\, u;\ y\!:\!\mathsf{Pos}_H\,(h\, r_1)) \\
\mid \mathsf{sup}\, u\,[w, h'];\ \mathsf{below}\, r_1\, x \triangleright\ (r_0\!:\!R_0\, u) + \\
\qquad\qquad\qquad ((r_1'\!:\!-r_1;\ z\!:\!\mathsf{Pos}_H\,(h'\, r_1')) + (x'\!:\!\mathsf{Pos}_H\, w - x))
\end{array}\right)
\end{array}$$

Remark—in the step where we split h at r_1, we did not first apply algebraic laws, permuting the patterns to move r_1 left of h; we could clearly have done so, at the cost of some obfuscation.

As one might expect, the hole is either at the top node or below it, whilst the positions explain the ways to avoid the hole. We illustrate the possibilities below, showing the components of the position information as named above.



Observe that if the hole is on top (r_1), another position must either be elsewhere on top (r'_1) or anywhere (y) in any subtree (r_2). If the hole is below (r_2), another position must be either on top (r_1), or anywhere (z) in a different subtree (r'_2), or in the same subtree as the hole but elsewhere (x').

We arrive at the same analysis when we expand $\mu H'$.

$$
\begin{aligned}
&\mu H'\\
&\qquad [\text{EXPAND } H'\rangle\\
=\;& \mu((\uparrow \partial_0 H[\phi H]) + (\uparrow \partial_1 H[\phi H]) \times \mathsf{P}_1)\\
&\qquad [\text{EXPAND } H,\phi\rangle\\
=\;& \mu\begin{pmatrix} \quad(\uparrow \partial_0(u \triangleright R_1\, u, R_2\, u)[w \triangleright \mathsf{Pos}_H\, w])\\ +\,(\uparrow \partial_1(u \triangleright R_1\, u, R_2\, u)[w \triangleright \mathsf{Pos}_H\, w]) \times \mathsf{P}_1\end{pmatrix}\\
&\qquad [\text{EXPAND } \partial_0,\partial_1\rangle\\
=\;& \mu\begin{pmatrix} \quad(\uparrow (u;\; r_0 : R_0\, u \triangleright -r_0, R_1\, u)[w \triangleright \mathsf{Pos}_H\, w])\\ +\,(\uparrow (u;\; r_1 : R_1\, u \triangleright R_0\, u, -r_1)[w \triangleright \mathsf{Pos}_H\, w]) \times \mathsf{P}_1\end{pmatrix}\\
&\qquad [\text{EXPAND } -[-],\uparrow,\mathsf{P}_1\rangle\\
=\;& \mu\begin{pmatrix} \quad u;\; r_0;\; h : R_1\, u \to \Phi \triangleright (-r_0) + (r : R_1\, u;\; \mathsf{Pos}_H\,(h\, r)), 0\\ +\begin{pmatrix} \quad u;\; r_1;\; h' : (-r_1) \to \Phi \triangleright R_0\, u + (r' : -r_1;\; \mathsf{Pos}_H\,(h'\, r')), 0\\ \times\, 1 \triangleright 0, 1\end{pmatrix}\end{pmatrix}\\
&\qquad [\text{EXPAND } \times\rangle\\
=\;& \mu\begin{pmatrix} \quad u;\; r_0;\; h \triangleright (-r_0) + (r;\; \mathsf{Pos}_H\,(h\, r)), 0\\ +\, u;\; r_1;\; h' \triangleright R_0\, u + (r' : -r_1; \mathsf{Pos}_H\,(h'\, r')), 1\end{pmatrix}\\
&\qquad [\text{EXPAND } +\rangle\\
=\;& \mu\begin{pmatrix} \quad \mathsf{inl}\,(u, r_0, h) \;\; \triangleright (-r_0) + (r;\; \mathsf{Pos}_H\,(h\, r)) \qquad\;\; , 0\\ \mid \mathsf{inr}\,(u, r_1, h') \triangleright R_0\, u + (r' : -r_1; \mathsf{Pos}_H\,(h'\, r')), 1\end{pmatrix}\\
&\qquad [\text{EXPAND } \mu\rangle\\
=\;& w' : \mathsf{W}\, U'\, R'_1 \triangleright \mathsf{Pos}_{(u' : U' \triangleright R'_0\, u', R'_1\, u')}\, w'
\end{aligned}
$$

where

U' contains the shape u of one node,
the hole r_0 or a step r_1 towards it,
the shapes h or h' of the subtrees where the hole is not

$$
U' = \begin{array}{ll} \quad (u : U;\; r_0 : R_0\, u;\; h : R_1\, u \to \Phi) & \textit{hole } r_0 \textit{ on top}\\ +\, (u : U;\; r_1 : R_1\, u;\; h' : (-r_1) \to \Phi) & \textit{hole below } r_1\end{array}
$$

R'_0 explains how to diverge from the hole's path at the top node

$$
\begin{array}{ll}
\quad\textit{if hole on top} & \textit{go elsewhere on top, or below}\\
R'_0\,(\mathsf{inl}\,(u, r_0, h)) & = (r'_0 : -r_0) + (r_1;\; y : \mathsf{Pos}_H\,(h\, r_1))\\
\quad\textit{if hole below} & \textit{go on top, or below to a different subtree}\\
R'_0\,(\mathsf{inr}\,(u, r_1, h')) & = (r_0 : R_0\, u) + (r'_1 : -r_1; z : \mathsf{Pos}_H\,(h'\, r'_1))
\end{array}
$$

R'_1 explains how to follow the hole's path one step below the top node

$$
\begin{array}{ll}
\quad\textit{if hole on top} & \textit{you can't follow its path below}\\
R'_1\,(\mathsf{inl}\,(u, r_0, h)) & = 0\\
\quad\textit{if hole below} & \textit{you can}\\
R'_1\,(\mathsf{inr}\,(u, r_1, h')) & = 1
\end{array}
$$

Where the shape of the derivative above comprised a whole shape $w : \Phi$ and a position within $\mathsf{Pos}_H\, w$, we now have an inductively defined 'shape context' $w' : \mathsf{W}\, U'\, R'_1$ representing both the path to the hole and the shape information surrounding it. For each node on the path, the latter comprises the local shape u and a function (h or h') giving shapes to the subtrees branching away at that point. A position is thus a path which diverges from the path to the hole at some point—either at the top node, or following the hole's path into a subtree r_1 and diverging later.

The following container isomorphism (σ, ψ) converts between these two representations. This recursive definition is terminating: σ and ψ^{-1} are structurally recursive on the position of the hole, whilst σ^{-1} and ψ are structurally recursive on the whole 'shape context'. Again, we are careful to keep names consistent with the diagram.

$$
\begin{array}{lcl}
\sigma\,(\mathsf{sup}\, u\, h, \mathsf{top}\, r_0) & \rightleftharpoons & \mathsf{sup}\,(\mathsf{inl}\,(u, r_0, h), ())\\
\quad \rhd \begin{cases} \mathsf{inl}\, r'_0 \\ \mathsf{inr}\,(r_1, y) \end{cases} & \rightleftharpoons & \quad \rhd\ \psi \begin{cases} \mathsf{top}\,(\mathsf{inl}\, r'_0) \\ \mathsf{top}\,(\mathsf{inr}\,(r_1, y)) \end{cases}\\
\sigma\,(\mathsf{sup}\, u\,[w, h'], \mathsf{below}\, r_1\, x) & \rightleftharpoons & \mathsf{sup}\,(\mathsf{inr}\,(u, r_1, h'))\,(\sigma\,(w, x))\\
\quad \rhd \begin{cases} \mathsf{inl}\, r_0 \\ \mathsf{inr}\,(\mathsf{inl}\,(r'_1, z)) \\ \mathsf{inr}\,(\mathsf{inr}\,(\psi\, x')) \end{cases} & \rightleftharpoons & \quad \rhd\ \psi \begin{cases} \mathsf{top}\,(\mathsf{inl}\, r_0) \\ \mathsf{top}\,(\mathsf{inr}(r'_1, z)) \\ \mathsf{below}\,()\, x' \end{cases}
\end{array}
$$

9 Conclusions

The present paper introduces and analyses a differential calculus of datatypes with clear applications in generic programming and constructive reasoning. While our conference paper (Abbott et al. (2003b)) and the work presented in Abbott (2003) presented derivatives in category-theoretic terms, the present paper gives an account using the language of constructive set theory. We employ pattern matching notation and container diagrams to provide an intuitive account of the constructions.

Already in the conference paper we discussed extending the underlying notion of datatypes, i.e. containers, to include quotients of positions to encompass type like the types of bags or multisets. Indeed, multisets resemble the exponential function, and remain the same under differentiation. Including quotients should make it possible to use the equivalent of Taylor's theorem to analyse datatypes. Due to reasons of time and space we do not develop this topic here but leave it to further work to present the development of quotient containers and their use in the differential calculus of datatypes.

Currently, we only use the specification of derivatives to derive the concrete implementation of differentiation for containers. We hope to be able to prove the laws of differentiation directly from the universal property, this generic approach would then also extend to quotient containers directly and would not need the prerequisite of decidability.

An obvious asymmetry in our current presentation is that we use dependent types in the explanation of data structures however, we do not analyse dependently typed structures. We hope to be able to extend our work in this direction which would also provide an important stepping stone towards analysing higher order types.

Finally, the constructions presented in this paper are ideal candidates for the development of a library within a dependently typed programming language whose correctness can be verified by type checking. We will embark on this project as soon as the tools it requires become available.

References

M. Abbott. *Categories of Containers*. PhD thesis, University of Leicester, 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, 2003a.

M. Abbott, T. Altenkirch, and N. Ghani. Representing nested inductive types using W-types. Submitted for publication, 2004a.

M. Abbott, T. Altenkirch, and N. Ghani. Representing strictly positive types. Draft, 2004b.

M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. Derivatives of containers. In *Typed Lambda Calculi and Applications, TLCA*, 2003b.

P. Aczel. On relating type theories and set theories. *Lecture Notes in Computer Science*, 1657:1–20, 1999.

T. Ehrhard and L. Regnier. The differential lambda-calculus. URL `http://iml.univ-mrs.fr/ftp/regnier/Articles.html`. Available electronically, 2001.

M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14th LICS Conf.*, pages 193–202. IEEE, Computer Society Press, 1999.

R. Hasegawa. Two applications of analytic functors. *Theoretical Comput. Sci.*, 272(1-2):112–175, 2002.

M. Hofmann. On the interpretation of type theory in locally cartesian closed catetories. In *Computer Science Logic*, volume 933 of *LNCS*, 1994.

M. Hofmann. Syntax and semantics of dependent types. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, volume 14, pages 79–130. Cambridge University Press, Cambridge, 1997.

P. Hoogendijk and O. de Moor. Container types categorically. *Journal of Functional Programming*, 10(2):191–225, 2000.

G. Huet. The zipper. *Journal of Functional Programming*, 7(5):549–554, 1997.

C. B. Jay and J. R. B. Cockett. Shapely types and shape polymorphism. In D. Sannella, editor, *Programming Languages and Systems - ESOP '94: 5th European Symposium on Programming, U.K., April 1994, Proceedings*, pages 302–316. Springer-Verlag, 1994.

A. Joyal. Foncteurs analytiques et espéces de structures. In *Combinatoire énumérative*, number 1234 in LNM, pages 126 – 159. 1986.

C. McBride. The derivative of a regular type is its type of one-hole contexts. URL `http://www.dur.ac.uk/c.t.mcbride/`. Available electronically, 2001.

I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104:189–218, 2000.