A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algoritm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algoritms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algoritm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algoritms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algoritm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algoritms
- How do we describe this in Categorical notation?

A short introdution

- In chapter 4 we've derived an AD algorithm that was generalized in figure 6 of the document
- With fully right-associated compositions this algoritm becomes a foward-mode AD and with fully left-associated becomes a reverse-mode AD
- We want to obtain generalized FAD and RAD algoritms
- How do we describe this in Categorical notation?

Converting morfisms

Given a category k we can represent its morfisms using the intent to left-compose with another morfism:
f :: a'k'b becomes ($\circ$ f) :: (b'k'r) $\rightarrow$ (a'k'r) where r is any object of k. If h is the morfism we'll compose with f then h is the continuation of f.
With this idea in mind we can derive a category based on it, creating a generalization of the RAD algoritm.

Instance deduction

We'll begin by creating a new data type:

$$\textbf{newtype}\ Cont\ (k, r)\ a\ b = Cont\ ((b'k'r) \to (a'k'r))$$

And then defining a functor from it:

$$cont :: Category\ k \Rightarrow (a\ '\mathrm{k}'\ b) \to Cont\ (k, r)\ a\ b$$
$$cont\ f = Cont\ (\circ f)$$

With this we can derive new categorical isntances:

Instance deduction

instance *Category k* $\Rightarrow$ *Category Cont* (k, r) **where**
  $id = Cont\ id$
  $Cont\ g \circ Cont\ f = Cont\ (f \circ g)$

instance *Monoidal k* $\Rightarrow$ *Monoidal Cont* (k, r) **where**
  $Conf\ f\ x\ Cont\ g = Cont\ (\bigtriangleup \circ (f\ x\ g) \circ unjoin)$

instance *Cartesian k* $\Rightarrow$ *Cartesian Cont* (k, r) **where**
  $exl = Cont\ (\bigtriangleup \circ inl); exr = Cont\ (\bigtriangleup \circ inr)$
  $dup = Cont\ (jam \circ unjoin)$

instance *Cocartesian k* $\Rightarrow$ *Cocartesian Cont* (k, r) **where**
  $inl = Cont\ (exl \circ unjoin); inr = Cont\ (exr \circ unjoin)$
  $jam = Cont\ (\bigtriangleup \circ dup)$

instance *Scalable k a* $\Rightarrow$ *Scalable Cont* (k, r) *a* **where**
  $scale\ s = Cont\ (scale\ s)$

A short introdution

Due to it's widespread use in ML we'll talk about a specific case of RAD: computing gradients(derivatives of functions with scalar codomains)
A vector space A over a scalar field has A $\multimap$ s as it's dual(i.e., the linear maps of the udnerlaying field of A are it's dual) This dual space is also a vector space and if A is finite in dimention they are isomorfic.
Each linear map in A $\multimap$ s can be represented in the form of dot u for some u :: A where

$$\textbf{class } HasDot\ (S)\ u \textbf{ where } dot :: u \rightarrow (u - o\ s)$$

$$\textbf{instance } HasDot\ (IR)\ IR \textbf{ where } dot = scale$$

$$\textbf{instance } (HasDot\ (S)\ a, HasDot\ (S)\ b) \Rightarrow HasDot\ (S)\ (a\ x\ b)$$

$$\textbf{where } dot\ (u, v) = dot\ u\ \nabla\ dot\ v$$

Instance deduction

Since $Cont_k^r$ works for any type/object r we can use it with the scalar field s.
The internal representation of $Cont_{\multimap}^s$ a b is (b $\multimap$ s) -> (a $\multimap$ s) which is isomorfic to (a -> b). With this in mind we can call this representation as the dual/opposite of k:

$$\textbf{newtype}\ Dual\ (K)\ a\ b = Dual\ (b' k' a)$$

Instance deduction

With this construction all we need to do to create dual representations of linear maps is to convert from $Cont_k^S$ to $Dual_k$ using a functor that we'll now derive:

$$
\begin{aligned}
& asDual :: (HasDot\ (S)\ a, HasDot\ (S)\ b) \Rightarrow \\
& \quad ((b \multimap s) \to (a \multimap s)) \to (b \multimap a) \\
& asDual\ (Cont\ f) = Dual\ (onDot\ f)
\end{aligned}
$$

where

$$
\begin{aligned}
& onDot :: (HasDot\ (S)\ a, HasDot\ (S)\ b) \Rightarrow \\
& \quad ((b \multimap s) \to (a \multimap s)) \to (b \multimap a) \\
& onDot\ f = dot \uparrow \{\texttt{"-1"}\} \circ f \circ dot
\end{aligned}
$$

Given this we can now derive our new categorical instances

Instance deduction

$$
\begin{aligned}
&\textbf{instance } \mathit{Category}\ k \Rightarrow \mathit{Category}\ \mathit{Dual}\ (k)\ \textbf{where}\\
&\quad \mathit{id} = \mathit{Dual}\ \mathit{id}\\
&\quad \mathit{Dual}\ g \circ \mathit{Dual}\ f = \mathit{Dual}\ (f \circ g)\\
&\textbf{instance } \mathit{Monoidal}\ k \Rightarrow \mathit{Monoidal}\ \mathit{Dual}\ (k)\ \textbf{where}\\
&\quad \mathit{Dual}\ f\ x\ \mathit{Dual}\ g = \mathit{Dual}\ (f\ x\ g)\\
&\textbf{instance } \mathit{Cartesian}\ k \Rightarrow \mathit{Cartesian}\ \mathit{Dual}\ (k)\ \textbf{where}\\
&\quad \mathit{exl} = \mathit{Dual}\ \mathit{inl}; \mathit{exr} = \mathit{Dual}\ \mathit{inr}; \mathit{dup} = \mathit{Dual}\ \mathit{jam}\\
&\textbf{instance } \mathit{Cocartesian}\ k \Rightarrow \mathit{CocartesianDual}\ (k)\ \textbf{where}\\
&\quad \mathit{inl} = \mathit{Dual}\ \mathit{exl}; \mathit{inr} = \mathit{Dual}\ \mathit{exr}; \mathit{jam} = \mathit{Dual}\ \mathit{dup}\\
&\textbf{instance } \mathit{Scalable}\ k \Rightarrow \mathit{Scalable}\ \mathit{Dual}\ (k)\ \textbf{where}\\
&\quad \mathit{scale}\ s = \mathit{Dual}\ (\mathit{scale}\ s)
\end{aligned}
$$

Final notes

- $\triangle$ *and* $\triangledown$ mutually dualize $(Dual\ f\ \triangle\ Dual\ g = Dual\ (f\ \triangledown\ g)$ *and* $Dual\ f\ \triangledown\ Dual\ g = Dual\ (f\ \triangle\ g))$
- Using the definition from chapter 8 we can determine that the duality of a matrix corresponds to it's transposition

Fowards-mode Automatic Differentiation(FAD)

We can use the same deductions we've done in Cont and Dual to derive a category with full right-side association, thus creating a generized FAD algorithm. This algorithm is far more apropriated for low dimention domains.

$$\textbf{newtype}\ \mathit{Begin}\ (k, r)\ a\ b = \mathit{Begin}\ ((r'k'a) \rightarrow (r'k'b))$$
$$\mathit{begin} :: \mathit{Category}\ k \Rightarrow (a\ '\mathtt{k}'\ \ b) \rightarrow \mathit{Begin}\ (k, r)\ a\ b$$
$$\mathit{begin}\ f = \mathit{Begin}\ (f \circ)$$

We can derive categorical instances from the functor above and we can choose r to be the scalar field s, noting that s $\multimap$ a is isomorfic to a.