The Derivative of a Functor

RALF HINZE

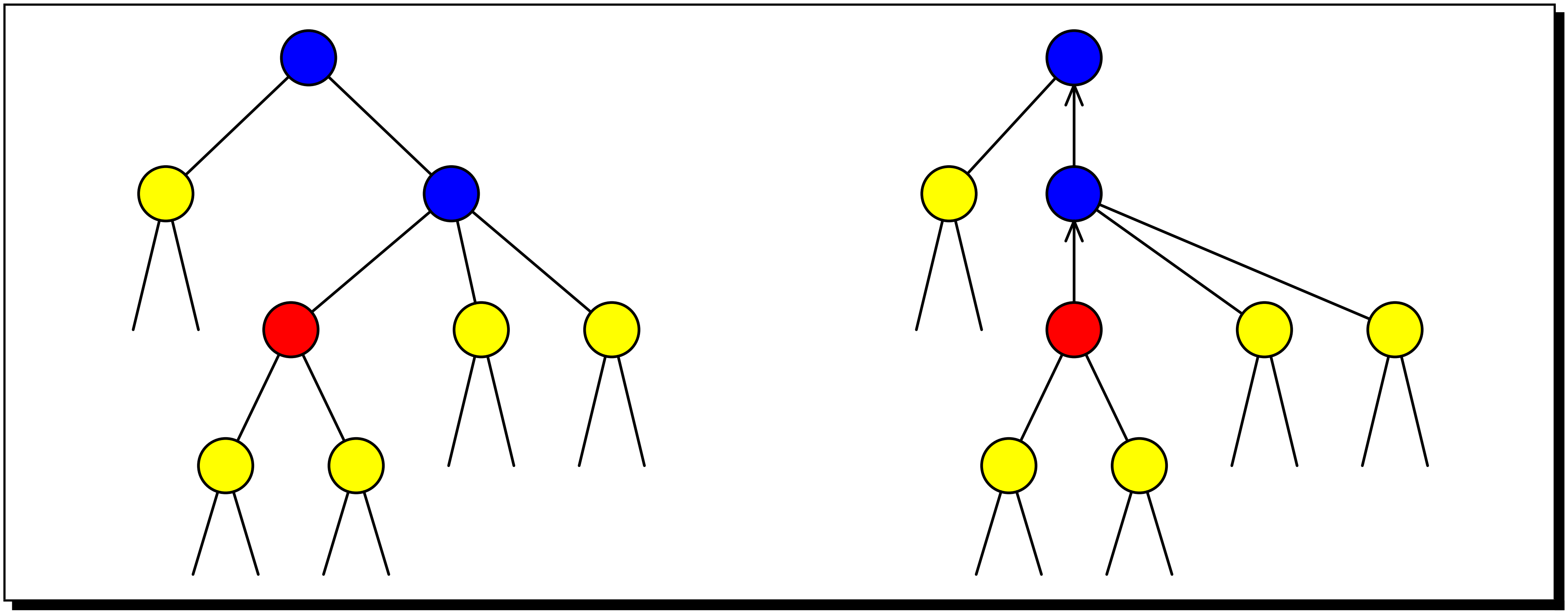
Institute of Information and Computing Sciences
Utrecht University
Email: `ralf@cs.uu.nl`
Homepage: `http://www.cs.uu.nl/~ralf/`

September, 2001

(Pick the slides at `.../~ralf/talks.html#T29`.)

Motivation

Task: represent a tree together with a focus of interest.



We are seeking a *generic definition*, that is, one that works for arbitrary (recursive) data types.

A concrete instance: 2-3 trees

$$
\begin{array}{lcl}
\textbf{data}\ \mathit{Tree23} & = & \mathit{empty} \\
 & \mid & \mathit{node2}\ \mathit{Tree23}\ \mathit{Int}\ \mathit{Tree23} \\
 & \mid & \mathit{node3}\ \mathit{Tree23}\ \mathit{Int}\ \mathit{Tree23}\ \mathit{Int}\ \mathit{Tree23}
\end{array}
$$

$$
\begin{array}{l}
\mathit{node2}\ (\mathit{node2}\ \mathit{empty}\ 1\ \mathit{empty}) \\
\qquad 2\ (\mathit{node3}\ (\mathit{node2}\ (\mathit{node2}\ \mathit{empty}\ 3\ \mathit{empty}) \\
\qquad\qquad\qquad\qquad 4\ (\mathit{node2}\ \mathit{empty}\ 5\ \mathit{empty})) \\
\qquad\qquad\quad 6\ (\mathit{node2}\ \mathit{empty}\ 7\ \mathit{empty}) \\
\qquad\qquad\quad 8\ (\mathit{node2}\ \mathit{empty}\ 9\ \mathit{empty}))
\end{array}
$$

A 2-3 tree with a focus of interest consists of the focused tree and a path leading to the root.

$$
\begin{array}{lcl}
\textbf{type}\ \mathit{Focus23} & = & (\mathit{Path23}, \mathit{Tree23}) \\
\textbf{data}\ \mathit{Path23} & = & \mathit{top} \mid \mathit{step}\ \mathit{Path23}\ \mathit{Seg23} \\
\textbf{data}\ \mathit{Seg23} & = & \mathit{node2}_1\ \bullet\ \mathit{Int}\ \mathit{Tree23} \\
 & \mid & \mathit{node2}_2\ \mathit{Tree23}\ \mathit{Int}\ \bullet \\
 & \mid & \mathit{node3}_1\ \bullet\ \mathit{Int}\ \mathit{Tree23}\ \mathit{Int}\ \mathit{Tree23} \\
 & \mid & \mathit{node3}_2\ \mathit{Tree23}\ \mathit{Int}\ \bullet\ \mathit{Int}\ \mathit{Tree23} \\
 & \mid & \mathit{node3}_3\ \mathit{Tree23}\ \mathit{Int}\ \mathit{Tree23}\ \mathit{Int}\ \bullet
\end{array}
$$

NB. $\mathit{Path23}$ is a snoc list of $\mathit{Seg23}$'s.

NB. $\bullet$ is the unit type (representing a hole).

```
(        -- up path
 step (step top
             (node2₂ (node2 empty 1 empty)
                     2 •))
      (node3₁ •
               6 (node2 empty 7 empty)
               8 (node2 empty 9 empty))
      -- focused tree
, node2 (node2 empty 3 empty)
        4 (node2 empty 5 empty)
)
```

Making recursive components explicit

We write the type $\mathit{Tree23}$ as a fixed point of a *functor*.

$$\begin{array}{lcl} \mathit{Tree23} & = & \mathit{Fix}\ \mathit{Base23} \\ \mathit{Base23} & = & \mathit{empty}\ (K\ 1) \\ & + & \mathit{node2}\ (\mathit{Id} \times \mathit{Int} \times \mathit{Id}) \\ & + & \mathit{node3}\ (\mathit{Id} \times \mathit{Int} \times \mathit{Id} \times \mathit{Int} \times \mathit{Id}) \end{array}$$

NB. $K\ T$ is the constant functor, Id is the identity functor, and '$+$' and '$\times$' denote lifted sums and pairs.

The fixed point operator, Fix, is given by

$$\mathbf{data}\ \mathit{Fix}\ F\ \ =\ \ \mathit{in}\{\mathit{out} :: F\ (\mathit{Fix}\ F)\}.$$

$$\begin{array}{rcl} \mathit{Focus23} & = & \mathit{Path23} \times \mathit{Tree23} \\ \mathit{Path23} & = & \mathit{top}\ (1) + \mathit{step}\ (\mathit{Path23} \times \mathit{Seg23}) \end{array}$$

$$\begin{array}{rcl} \mathit{Seg23} & = & \mathit{Base23}'\ \mathit{Tree23} \\ \mathit{Base23}' & = & \mathit{node2}_1\ (K \bullet \times K\ \mathit{Int} \times \mathit{Id}) \\ & + & \mathit{node2}_2\ (\mathit{Id} \times K\ \mathit{Int} \times K \bullet) \\ & + & \mathit{node3}_1\ (K \bullet \times K\ \mathit{Int} \times \mathit{Id} \times K\ \mathit{Int} \times \mathit{Id}) \\ & + & \mathit{node3}_2\ (\mathit{Id} \times K\ \mathit{Int} \times \bullet \times K\ \mathit{Int} \times \mathit{Id}) \\ & + & \mathit{node3}_3\ (\mathit{Id} \times K\ \mathit{Int} \times \mathit{Id} \times K\ \mathit{Int} \times K \bullet) \end{array}$$

Generic paths and segments

Let T be the fixed point of F, that is, $T = \mathit{Fix}\ F$. We parameterize the generic types by the base functor F.

$$
\begin{aligned}
\mathit{Focus}\ F &= \mathit{Path}\ F \times \mathit{Fix}\ F \\
\mathit{Path}\ F &= \mathit{top}\ (1) + \mathit{step}\ (\mathit{Path}\ F \times \mathit{Seg}\ F) \\
\mathit{Seg}\ F &= F'\ (\mathit{Fix}\ F)
\end{aligned}
$$

☞ Now, what is the relationsship between F and F'?

The derivative of a functor

The functor F' is the derivative of F. We define F' by induction on the structure of F.

$$
\begin{array}{lcl}
(K\ C)' & = & K\ 0 \\
Id' & = & K\ 1 \\
(F_1 + F_2)' & = & F_1' + F_2' \\
(F_1 \times F_2)' & = & F_1' \times F_2 + F_1 \times F_2'
\end{array}
$$

NB. Recall that $\bullet = 1$.

☞ The observation that a *one-point context* corresponds to the derivative of a functor is due to McBride (the definition, however, was given independently by Hinze/Jeuring).

Examples

$$
\begin{array}{lcl}
(Id + Id)' & = & K\ 1 + K\ 1 \\
(K\ n \times Id)' & \cong & K\ n \\[1ex]
(Id \times Id)' & = & K\ 1 \times Id + Id \times K\ 1 \\
(Id^n)' & \cong & K\ n \times Id^{n-1} \\[1ex]
List' & = & K\ 0 + (K\ 1 \times List + Id \times List') \\
List' & \cong & List \times List
\end{array}
$$

☞ The derivative of the list functor is a pair of lists (the prefix and the suffix of the hole).

The chain rule

From high school math we all know and love the *chain rule*:

$$(F \cdot G)' \cong F' \cdot G \times G'.$$

Proof of the chain rule

The proof proceeds by fixed point induction on F_1.

Case $F = F_1 \times F_2$:

$$
\begin{array}{cl}
 & ((F_1 \times F_2) \cdot G)' \\
= & \quad \{ \text{ composition distributes leftward through `}\times\text{' } \} \\
 & (F_1 \cdot G \times F_2 \cdot G)' \\
= & \quad \{ \text{ product rule } \} \\
 & (F_1 \cdot G)' \times F_2 \cdot G + F_1 \cdot G \times (F_2 \cdot G)' \\
= & \quad \{ \text{ ex hypothesi } \} \\
 & F_1' \cdot G \times G' \times F_2 \cdot G + F_1 \cdot G \times F_2' \cdot G \times G'
\end{array}
$$

$$
\begin{array}{ll}
& F_1' \cdot G \times G' \times F_2 \cdot G + F_1 \cdot G \times F_2' \cdot G \times G' \\
\cong & \quad \{ \text{swapping: } A \times B \cong B \times A \} \\
& F_1' \cdot G \times F_2 \cdot G \times G' + F_1 \cdot G \times F_2' \cdot G \times G' \\
= & \quad \{ \text{`}\times\text{' distributes through `}+\text{'} \} \\
& (F_1' \cdot G \times F_2 \cdot G + F_1 \cdot G \times F_2' \cdot G) \times G' \\
= & \quad \{ \text{composition distributes leftward through `}+\text{' and `}\times\text{'} \} \\
& (F_1' \times F_2 + F_1 \times F_2') \cdot G \times G' \\
= & \quad \{ \text{product rule} \} \\
& (F_1 \times F_2)' \cdot G \times G'
\end{array}
$$

Examples

$$
\begin{array}{lcl}
(List \cdot List)' & \cong & List' \cdot List \times List' \\
(List \cdot List)' & \cong & List^2 \cdot List \times List^2
\end{array}
$$

The chain rule is convenient for calculating the derivatives of so-called *nested data types*.

$$
\begin{array}{lcl}
Pair & = & Id \times Id \\
Perfect & = & Id + Perfect \cdot Pair \\
Perfect' & \cong & K1 + Perfect' \cdot Pair \times K\ 2 \times Id \\
Perfect' & \cong & K1 + Perfect' \cdot Pair \times Id + Perfect' \cdot Pair \times Id
\end{array}
$$

Operations

Moving up a 2-3 tree:

$$
\begin{array}{lcl}
up & :: & Focus23 \rightarrow Focus23 \\
up\ (top, t) & = & (top, t) \\
up\ (step\ p\ s, t) & = & (p, plugin\ s\ t) \\
\\
plugin & :: & Seg23 \rightarrow Tree23 \rightarrow Tree23 \\
plugin\ (node2_1\ ()\ a\ r)\ t & = & node2\ t\ a\ r \\
plugin\ (node2_2\ l\ a\ ())\ t & = & node2\ l\ a\ t \\
plugin\ (node3_1\ ()\ a\ m\ b\ r)\ t & = & node3\ t\ a\ m\ b\ r \\
plugin\ (node3_2\ l\ a\ ()\ b\ r)\ t & = & node3\ l\ a\ t\ b\ r \\
plugin\ (node3_3\ l\ a\ m\ b\ ())\ t & = & node3\ l\ a\ m\ b\ t
\end{array}
$$

NB. Recall that $() = \bullet$.

Generic operations

$$
\begin{array}{lcl}
\mathit{up}_F & :: & \mathit{Focus}\ F \rightarrow \mathit{Focus}\ F \\
\mathit{up}_F\ (\mathit{top}, t) & = & (\mathit{top}, t) \\
\mathit{up}_F\ (\mathit{step}\ (p, s), t) & = & (p, \mathit{in}\ (\mathit{plugin}_F\ (s, t))) \\
\mathit{plugin}_F & :: & \forall A\,.\,F'\ A \times A \rightarrow F\ A \\
\mathit{plugin}_{\mathit{Id}}\ (\bullet, t) & = & t \\
\mathit{plugin}_{F_1+F_2}\ (\mathit{inl}\ s_1, t) & = & \mathit{inl}\ (\mathit{plugin}_{F_1}\ (s_1, t)) \\
\mathit{plugin}_{F_1+F_2}\ (\mathit{inr}\ s_2, t) & = & \mathit{inr}\ (\mathit{plugin}_{F_2}\ (s_2, t)) \\
\mathit{plugin}_{F_1\times F_2}\ (\mathit{inl}\ (s, r), t) & = & (\mathit{plugin}_{F_1}\ (s, t), r) \\
\mathit{plugin}_{F_1\times F_2}\ (\mathit{inr}\ (l, s), t) & = & (l, \mathit{plugin}_{F_2}\ (s, t)).
\end{array}
$$

NB. We need not define $\mathit{plugin}_{K\ C}$ as $(K\ C)' = K\ 0$.

Future work

- ✗ The definition of $(-)'$ can be easily generalized to arbitrary types of arbitrary kinds.

- ✗ *Open problem:* generalization of the chain rule using logical relations.