Zippers and Derivatives

Shuwei Hu
Supervised by: Manuel Eberl

December 11, 2017

Outline

Overview

- Zipper = context type, which helps moving through and "modifying" a functional data structure [1]

Overview

- Zipper = context type, which helps moving through and "modifying" a functional data structure [1]
- Deriving context types vs. differentiating real-valued functions [2]

Overview

- Zipper = context type, which helps moving through and "modifying" a functional data structure [1]
- Deriving context types vs. differentiating real-valued functions [2]
- Relation with combinatorial species [3]

Outline

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor
- How to define a "pointer" `p` into a list `l`, supporting:

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor
- How to define a “pointer” `p` into a list `l`, supporting:
  - `p = begin(l)`

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor
- How to define a "pointer" `p` into a list `l`, supporting:
  - `p = begin(l)`
  - `prev(p)`

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor
- How to define a "pointer" `p` into a list `l`, supporting:
  - `p = begin(l)`
  - `prev(p)`
  - `next(p)`

Context for List I

- `datatype 'a list = Nil | Cons 'a ('a list)`
- Suppose we are implementing a functional text editor
- How to define a "pointer" `p` into a list `l`, supporting:
  - `p = begin(l)`
  - `prev(p)`
  - `next(p)`
  - `*p := a`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
  - `(...  <- x <- x <- y) (z) (z -> ...)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
  - `(...  <- x <- x <- y) (z) (z -> ...)`
- `assign (xs, _, zs) y = (xs, y, zs)`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(... <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(... <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
  - `(... <- x <- x <- y) (z) (z -> ...)`
- `assign (xs, _, zs) y = (xs, y, zs)`
- `reconstruct (xs, y, zs) = rev xs @ [y] @ zs`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
    - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
    - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
    - `(...  <- x <- x <- y) (z) (z -> ...)`
- `assign (xs, _, zs) y = (xs, y, zs)`
- `reconstruct (xs, y, zs) = rev xs @ [y] @ zs`
- Equivalent definition:

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
  - `(...  <- x <- x <- y) (z) (z -> ...)`
- `assign (xs, _, zs) y = (xs, y, zs)`
- `reconstruct (xs, y, zs) = rev xs @ [y] @ zs`
- Equivalent definition:
  - `'a list_pointer = 'a * 'a list_context`

Context for List II

- `'a list_pointer = 'a list * 'a * 'a list`
  - `(...  <- x <- x) (y) (z -> z -> ...)`
- `begin (Cons x xs) = (Nil, x, xs)`
- `prev (x#xs, y, zs) = (xs, x, y#zs)`
  - `(...  <- x) (x) (y -> z -> z -> ...)`
- `next (xs, y, z#zs) = (y#xs, z, zs)`
  - `(...  <- x <- x <- y) (z) (z -> ...)`
- `assign (xs, _, zs) y = (xs, y, zs)`
- `reconstruct (xs, y, zs) = rev xs @ [y] @ zs`
- Equivalent definition:
  - `'a list_pointer = 'a * 'a list_context`
  - `'a list_context = 'a list * 'a list`

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                     * 'a btree_ancestors
  ```
  ```

Context for Binary Tree I

- ```
  'a btree = Leaf | Node 'a ('a btree) ('a btree)
  ```
- ```
  'a btree_pointer = 'a * 'a btree_context
  ```
- ```
  'a btree_context = 'a btree * 'a btree
                     * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
       Top
     | IsLeft  'a ('a btree) ('a btree_ancestors)
     | IsRight 'a ('a btree) ('a btree_ancestors)
  ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                     * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
      Top
    | IsLeft  'a ('a btree) ('a btree_ancestors)
    | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))
    ```
    ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                     * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
       Top
     | IsLeft  'a ('a btree) ('a btree_ancestors)
     | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))
    ```
    ```
    (Node 0 (Node 1 Leaf
                    (Node 2 Leaf
                            Leaf))
            (Node 3 Leaf
                    Leaf))
    ```
    ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                     * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
      Top
    | IsLeft  'a ('a btree) ('a btree_ancestors)
    | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))
    ```

    ```
    (Node 0 (Node 1 Leaf
                    (Node 2 Leaf
                            Leaf))
            (Node 3 Leaf
                    Leaf))
    ```
    ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                   * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
       Top
     | IsLeft  'a ('a btree) ('a btree_ancestors)
     | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))

    (Node 0 (Node 1 Leaf
                    (Node 2 Leaf
                            Leaf))
            (Node 3 Leaf
                    Leaf))
    ```
    ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                   * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
      Top
    | IsLeft  'a ('a btree) ('a btree_ancestors)
    | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))
    ```

    ```
    (Node 0 (Node 1 Leaf
                    (Node 2 Leaf
                            Leaf))
            (Node 3 Leaf
                    Leaf))
    ```
    ```

Context for Binary Tree I

- `'a btree = Leaf | Node 'a ('a btree) ('a btree)`
- `'a btree_pointer = 'a * 'a btree_context`
- ```
  'a btree_context = 'a btree * 'a btree
                   * 'a btree_ancestors
  ```
- ```
  'a btree_ancestors =
      Top
    | IsLeft  'a ('a btree) ('a btree_ancestors)
    | IsRight 'a ('a btree) ('a btree_ancestors)
  ```
  - ```
    (2, Leaf, Leaf,
     IsRight 1 Leaf (IsLeft 0 (Node 3 Leaf Leaf) Top))

    (Node 0 (Node 1 Leaf
                    (Node 2 Leaf
                            Leaf))
            (Node 3 Leaf
                    Leaf))
    ```
    ```

Context for Binary Tree II

- `up`, `down`, `left`, `right` for `btree_pointer`:

Context for Binary Tree II

- `up`, `down`, `left`, `right` for `btree_pointer`:
- ```
    up (a, (lc, rc, IsLeft p r anc))
     = (p, (Node a lc rc, r, anc))
  | up (a, (lc, rc, IsRight p l anc))
     = (p, (l, Node a lc rc, anc))
  ```
  ```

Context for Binary Tree II

- `up`, `down`, `left`, `right` for `btree_pointer`:

- ```
    up (a, (lc, rc, IsLeft p r anc))
     = (p, (Node a lc rc, r, anc))
  | up (a, (lc, rc, IsRight p l anc))
     = (p, (l, Node a lc rc, anc))
  ```

- ```
    left (a, (llc, lrc, IsRight p (Node b rlc rrc) anc))
       = (b, (rlc, rrc, IsLeft  p (Node a llc lrc) anc))
  ```

Context for Binary Tree II

- `up`, `down`, `left`, `right` for `btree_pointer`:
- ```
    up (a, (lc, rc, IsLeft p r anc))
     = (p, (Node a lc rc, r, anc))
  | up (a, (lc, rc, IsRight p l anc))
     = (p, (l, Node a lc rc, anc))
  ```
- ```
    left (a, (llc, lrc, IsRight p (Node b rlc rrc) anc))
       = (b, (rlc, rrc, IsLeft  p (Node a llc lrc) anc))
  ```
- `down` and `right` are defined similarly

Context for Binary Tree II

- `up`, `down`, `left`, `right` for `btree_pointer`:
- ```
    up (a, (lc, rc, IsLeft p r anc))
     = (p, (Node a lc rc, r, anc))
  | up (a, (lc, rc, IsRight p l anc))
     = (p, (l, Node a lc rc, anc))
  ```
- ```
    left (a, (llc, lrc, IsRight p (Node b rlc rrc) anc))
       = (b, (rlc, rrc, IsLeft  p (Node a llc lrc) anc))
  ```
- `down` and `right` are defined similarly
- Simpler definition:

```
'a btree_pointer = 'a * 'a bree_context
'a btree_context = 'a btree * 'a btree
                 * (bool * 'a * 'a btree) list
```

Context for Ordered Tree

- `'a tree = Leaf | Node 'a ('a tree list)`

Context for Ordered Tree

- `'a tree = Leaf | Node 'a ('a tree list)`

- `'a tree_pointer = 'a * 'a tree_context`

Context for Ordered Tree

- `'a tree = Leaf | Node 'a ('a tree list)`
- `'a tree_pointer = 'a * 'a tree_context`
- `'a tree_context = 'a tree list * 'a tree_ancestors`

Context for Ordered Tree

- `'a tree = Leaf | Node 'a ('a tree list)`
- `'a tree_pointer = 'a * 'a tree_context`
- `'a tree_context = 'a tree list * 'a tree_ancestors`
- ```
  'a tree_ancestors =
      Top
    | IsChild ('a tree list) 'a ('a tree list)
              ('a tree_ancestors)
  ```
  ```

Context for Ordered Tree

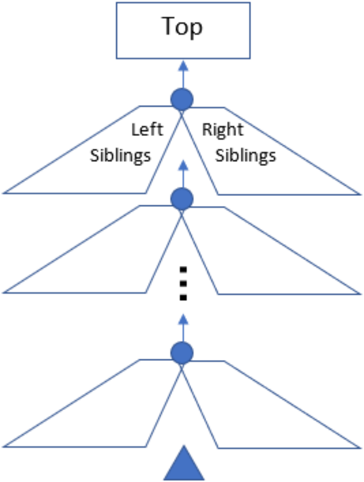
- `'a tree = Leaf | Node 'a ('a tree list)`
- `'a tree_pointer = 'a * 'a tree_context`
- `'a tree_context = 'a tree list * 'a tree_ancestors`
- ```
  'a tree_ancestors =
      Top
    | IsChild ('a tree list) 'a ('a tree list)
              ('a tree_ancestors)
  ```
- `up`, `down` `left`, `right` here are similar as for `btree_pointer`
  ```

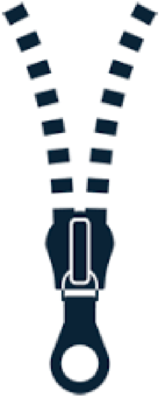
Context for Ordered Tree

- `'a tree = Leaf | Node 'a ('a tree list)`
- `'a tree_pointer = 'a * 'a tree_context`
- `'a tree_context = 'a tree list * 'a tree_ancestors`
- ```
  'a tree_ancestors =
       Top
     | IsChild ('a tree list) 'a ('a tree list)
               ('a tree_ancestors)
  ```
- `up`, `down left`, `right` here are similar as for `btree_pointer`
- Simpler definition:

```
'a tree_context = 'a tree list
            * ('a tree list * 'a * 'a tree list) list
```
```

Zipper![1]





[1] Image source: www.pacifictrimming.com

Huet's Zipper [1]

- For ordered trees with payload only on leaves:

Huet's Zipper [1]

- For ordered trees with payload only on leaves:
  - `'a ltree = Leaf 'a | Node ~~'a~~ ('a ltree list)`

Huet's Zipper [1]

- For ordered trees with payload only on leaves:
  - `'a ltree = Leaf 'a | Node` ~~'a~~ `('a ltree list)`
- Focus on a subtree instead of an element

Huet's Zipper [1]

- For ordered trees with payload only on leaves:
  - `'a ltree = Leaf 'a | Node ~~'a~~ ('a ltree list)`
- Focus on a subtree instead of an element
  - `'a ltree_pointer = ~~'a~~ 'a ltree * 'a ltree_context`

Huet's Zipper [1]

- For ordered trees with payload only on leaves:
  - `'a ltree = Leaf 'a | Node ~~'a~~ ('a ltree list)`
- Focus on a subtree instead of an element
  - `'a ltree_pointer = ~~'a~~ 'a ltree * 'a ltree_context`
  - `'a ltree_context = ~~'a ltree list *~~ 'a ltree_ancestors`

Huet's Zipper [1]

- For ordered trees with payload only on leaves:
  - `'a ltree = Leaf 'a | Node ~~'a~~ ('a ltree list)`
- Focus on a subtree instead of an element
  - `'a ltree_pointer = ~~'a~~ 'a ltree * 'a ltree_context`
  - `'a ltree_context = ~~'a ltree list *~~ 'a ltree_ancestors`
  -
    ```
    'a ltree_ancestors =
         Top
       | IsChild ('a ltree list) 'a ('a ltree list)
                 ('a ltree_ancestors)
    ```

Outline

Context Examples Recap

- ```
  'a list = unit + 'a * 'a list
  'a list_context = 'a list * 'a list
  ```
  ```

Context Examples Recap

- ```
  'a list = unit + 'a * 'a list
  'a list_context = 'a list * 'a list
  ```

- ```
  'a btree = unit + 'a * 'a btree * 'a btree
  'a btree_context = 'a btree * 'a btree
            * (bool * 'a * 'a btree) list
  ```

Context Examples Recap

- ```
  'a list = unit + 'a * 'a list
  'a list_context = 'a list * 'a list
  ```

- ```
  'a btree = unit + 'a * 'a btree * 'a btree
  'a btree_context = 'a btree * 'a btree
            * (bool * 'a * 'a btree) list
  ```

- ```
  'a tree = unit + 'a * a tree list
  'a tree_context = 'a tree list
            * ('a tree list * 'a * 'a tree list) list
  ```
  ```

Context Examples Recap

- ```
  'a list = unit + 'a * 'a list
  'a list_context = 'a list * 'a list
  ```
- ```
  'a btree = unit + 'a * 'a btree * 'a btree
  'a btree_context = 'a btree * 'a btree
            * (bool * 'a * 'a btree) list
  ```
- ```
  'a tree = unit + 'a * a tree list
  'a tree_context = 'a tree list
            * ('a tree list * 'a * 'a tree list) list
  ```
- Math-ly notation, e.g. $\mathbf{list}(a) = 1 + a \times \mathbf{list}(a)$
  ```

Context Examples Recap

- ```
  'a list = unit + 'a * 'a list
  'a list_context = 'a list * 'a list
  ```
- ```
  'a btree = unit + 'a * 'a btree * 'a btree
  'a btree_context = 'a btree * 'a btree
            * (bool * 'a * 'a btree) list
  ```
- ```
  'a tree = unit + 'a * a tree list
  'a tree_context = 'a tree list
            * ('a tree list * 'a * 'a tree list) list
  ```
- Math-ly notation, e.g. $\textbf{list}(a) = 1 + a \times \textbf{list}(a)$
- Context for an arbitrary algebraic data type?
  ```

Context of Basic Types

- Note the context of type a inside type T by $C[a](T)$

Context of Basic Types

- Note the context of type a inside type T by $C[a](T)$
  - e.g. $C[a](\textbf{list}(a)) = \textbf{list\_context}(a) = \textbf{list}(a) \times \textbf{list}(a)$

Context of Basic Types

- Note the context of type a inside type T by $C[a](T)$
  - e.g. $C[a](\textbf{list}(a)) = \textbf{list\_context}(a) = \textbf{list}(a) \times \textbf{list}(a)$
- Context of type a inside type 1 (i.e. `unit`): impossible!

Context of Basic Types

- Note the context of type a inside type T by $C[a](T)$
  - e.g. $C[a](\textbf{list}(a)) = \textbf{list\_context}(a) = \textbf{list}(a) \times \textbf{list}(a)$
- Context of type a inside type 1 (i.e. `unit`): impossible!
  - $C[a](1) = 0$

Context of Basic Types

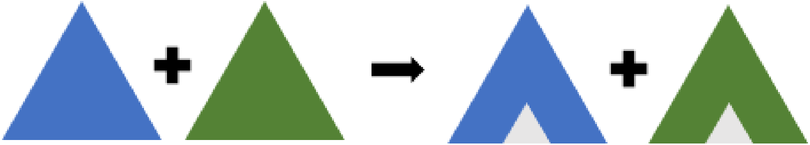
- Note the context of type a inside type T by $C[a](T)$
  - e.g. $C[a](\textbf{list}(a)) = \textbf{list\_context}(a) = \textbf{list}(a) \times \textbf{list}(a)$
- Context of type a inside type 1 (i.e. `unit`): impossible!
  - $C[a](1) = 0$
- Context of type a inside type a: dummy unit

Context of Basic Types

- Note the context of type a inside type T by $C[a](T)$
  - e.g. $C[a](\textbf{list}(a)) = \textbf{list\_context}(a) = \textbf{list}(a) \times \textbf{list}(a)$
- Context of type a inside type 1 (i.e. `unit`): impossible!
  - $C[a](1) = 0$
- Context of type a inside type a: dummy unit
  - $C[a](a) = 1$

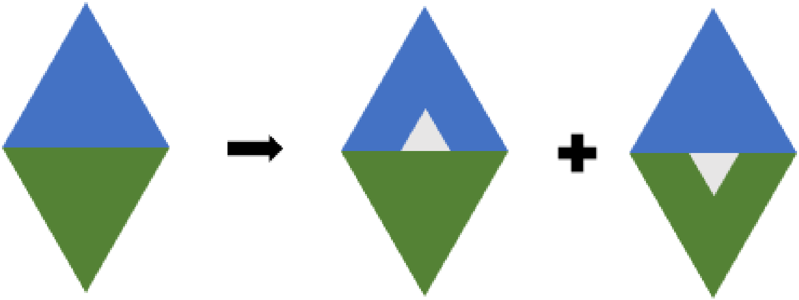
Context of Sum Type

- Inside $T_1 + T_2$, type a occurs in either of them
  - $C[a](T_1 + T_2) = C[a](T_1) + C[a](T_2)$



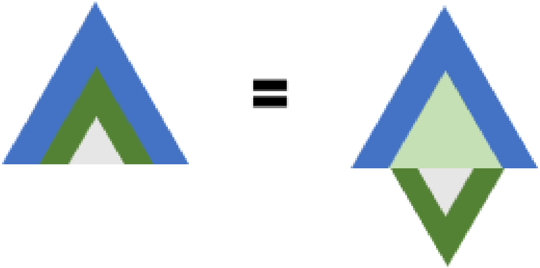
Context of Product Type

- Inside $T_1 \times T_2$, type a occurs in one of them, while the other must be carried in the context
  - $C[a](T_1 \times T_2) = C[a](T_1) \times T_2 + T_1 \times C[a](T_2)$ content



Context of Composed Type

- Inside composed type $T(U(a))$, type a occurs in one of U, which resides somewhere in T
  - $C[a](T(U(a))) = C[b](T(b))|_{b=U(a)} \times C[a](U(a))$



Context as Derivative

- Rules for context:
  - $C[a](1) = 0$
  - $C[a](a) = 1$
  - $C[a](T_1 + T_2) = C[a](T_1) + C[a](T_2)$
  - $C[a](T_1 \times T_2) = C[a](T_1) \times T_2 + T_1 \times C[a](T_2)$
  - $C[a](T(U(a))) = C[b](T(b))|_{b=U(a)} \times C[a](U(a))$
- Rules for derivative:
  - $\frac{\partial}{\partial x} c = 0$
  - $\frac{\partial}{\partial x} x = 1$
  - $\frac{\partial}{\partial x}(f + g) = \frac{\partial}{\partial x} f + \frac{\partial}{\partial x} g$
  - $\frac{\partial}{\partial x}(f \cdot g) = \frac{\partial}{\partial x} f \cdot g + f \cdot \frac{\partial}{\partial x} g$
  - $\frac{\partial}{\partial x}(f \circ g)|_{x_0} = \frac{\partial}{\partial u} f|_{u=g(x_0)} \cdot \frac{\partial}{\partial x} g|_{x=x_0}$
- $\frac{\partial}{\partial \mathrm{a}}(T) \triangleq C[a](T)$

Context of List, Revisit

$$\frac{\partial}{\partial \mathsf{a}}\,\textbf{list}(a) = \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(a))$$

Context of List, Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \textbf{list}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \textbf{list}(a))
\end{aligned}
$$

Context of List, Revisit

$$\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \,\textbf{list}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \textbf{list}(a)) \\
&= \textbf{list}(a) + a \times \frac{\partial}{\partial \mathsf{a}} \,\textbf{list}(a)
\end{aligned}$$

Context of List, Revisit

$$\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{list}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{list}(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{list}(a)) \\
&= \mathbf{list}(a) + a \times \frac{\partial}{\partial \mathsf{a}} \mathbf{list}(a) \\
\frac{\partial}{\partial \mathsf{a}} \mathbf{list}(a) &= \mathbf{list}(a) \times \mathbf{list}(a)
\end{aligned}$$

Context of List, Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \textbf{list}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \textbf{list}(a)) \\
&= \textbf{list}(a) + a \times \frac{\partial}{\partial \mathsf{a}} \textbf{list}(a) \\
\frac{\partial}{\partial \mathsf{a}} \textbf{list}(a) &= \textbf{list}(a) \times \textbf{list}(a) \\
&= \textbf{list\_context}(a)
\end{aligned}
$$

Binary Tree Context Revisit

$$\frac{\partial}{\partial \mathsf{a}} \textbf{btree}(a) = \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{btree}^2(a))$$

Binary Tree Context Revisit

$$\begin{aligned}\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{btree}^2(a)) \\ &= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{btree}^2(a))\end{aligned}$$

Binary Tree Context Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{btree}^2(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{btree}^2(a)) \\
&= \mathbf{btree}^2(a) + a \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}^2(a)
\end{aligned}
$$

Binary Tree Context Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{btree}^2(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{btree}^2(a)) \\
&= \mathbf{btree}^2(a) + a \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}^2(a) \\
&= \mathbf{btree}^2(a) + a \times 2 \times \mathbf{btree}(a) \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a)
\end{aligned}
$$

Binary Tree Context Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{btree}^2(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{btree}^2(a)) \\
&= \mathbf{btree}^2(a) + a \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}^2(a) \\
&= \mathbf{btree}^2(a) + a \times 2 \times \mathbf{btree}(a) \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) \\
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \mathbf{btree}^2(a) \times \mathbf{list}(2 \times a \times \mathbf{btree}(a))
\end{aligned}
$$

Binary Tree Context Revisit

$$\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{btree}^2(a)) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{btree}^2(a)) \\
&= \mathbf{btree}^2(a) + a \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}^2(a) \\
&= \mathbf{btree}^2(a) + a \times 2 \times \mathbf{btree}(a) \times \frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) \\
\frac{\partial}{\partial \mathsf{a}} \mathbf{btree}(a) &= \mathbf{btree}^2(a) \times \mathbf{list}(2 \times a \times \mathbf{btree}(a)) \\
&= \mathbf{btree_context}(a)
\end{aligned}$$

Context of Tree, Revisit

$$\frac{\partial}{\partial \mathsf{a}} \textbf{tree}(a) = \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(\textbf{tree}(a)))$$

Context of Tree, Revisit

$$\begin{aligned}\frac{\partial}{\partial \mathsf{a}} \mathbf{tree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{list}(\mathbf{tree}(a))) \\ &= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{list}(\mathbf{tree}(a)))\end{aligned}$$

Context of Tree, Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \textbf{tree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(\textbf{tree}(a))) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \textbf{list}(\textbf{tree}(a))) \\
&= \textbf{list}(\textbf{tree}(a)) + a \times \frac{\partial}{\partial \mathsf{a}}(\textbf{list}(\textbf{tree}(a)))
\end{aligned}
$$

Context of Tree, Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{tree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{list}(\mathbf{tree}(a))) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{list}(\mathbf{tree}(a))) \\
&= \mathbf{list}(\mathbf{tree}(a)) + a \times \frac{\partial}{\partial \mathsf{a}}(\mathbf{list}(\mathbf{tree}(a))) \\
&= \mathbf{list}(\mathbf{tree}(a)) + a \times \mathbf{list}^2(\mathbf{tree}(a)) \times \frac{\partial}{\partial \mathsf{a}}(\mathbf{tree}(a))
\end{aligned}
$$

Context of Tree, Revisit

$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \mathbf{tree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \mathbf{list}(\mathbf{tree}(a))) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \mathbf{list}(\mathbf{tree}(a))) \\
&= \mathbf{list}(\mathbf{tree}(a)) + a \times \frac{\partial}{\partial \mathsf{a}}(\mathbf{list}(\mathbf{tree}(a))) \\
&= \mathbf{list}(\mathbf{tree}(a)) + a \times \mathbf{list}^2(\mathbf{tree}(a)) \times \frac{\partial}{\partial \mathsf{a}}(\mathbf{tree}(a)) \\
\frac{\partial}{\partial \mathsf{a}} \mathbf{tree}(a) &= \mathbf{list}(\mathbf{tree}(a)) \times \mathbf{list}(a \times \mathbf{list}^2(\mathbf{tree}(a)))
\end{aligned}
$$

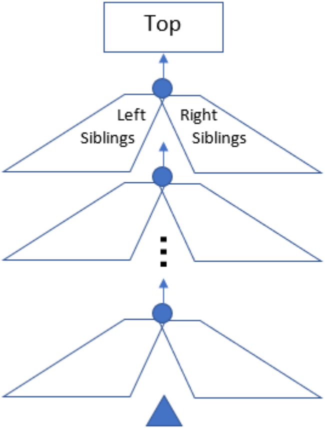
Context of Tree, Revisit

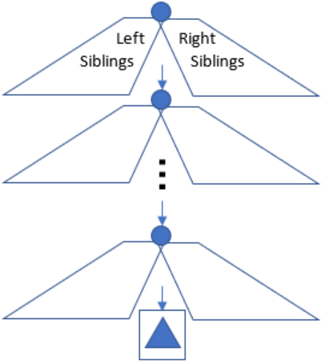
$$
\begin{aligned}
\frac{\partial}{\partial \mathsf{a}} \textbf{tree}(a) &= \frac{\partial}{\partial \mathsf{a}}(1 + a \times \textbf{list}(\textbf{tree}(a))) \\
&= 0 + \frac{\partial}{\partial \mathsf{a}}(a \times \textbf{list}(\textbf{tree}(a))) \\
&= \textbf{list}(\textbf{tree}(a)) + a \times \frac{\partial}{\partial \mathsf{a}}(\textbf{list}(\textbf{tree}(a))) \\
&= \textbf{list}(\textbf{tree}(a)) + a \times \textbf{list}^2(\textbf{tree}(a)) \times \frac{\partial}{\partial \mathsf{a}}(\textbf{tree}(a)) \\
\frac{\partial}{\partial \mathsf{a}} \textbf{tree}(a) &= \textbf{list}(\textbf{tree}(a)) \times \textbf{list}(a \times \textbf{list}^2(\textbf{tree}(a))) \\
&= \textbf{tree\_context}(a)
\end{aligned}
$$

Huet's Zipper, Revisit

- $\mathbf{ltree}(a) = a + \mathbf{list}(\mathbf{ltree}(a))$
- Differentiating against non-basic type is a bit tricky
- $\frac{\partial}{\partial\,\mathbf{ltree}}(\mathbf{ltree}) = 1$?
- $\frac{\partial}{\partial\,\mathbf{ltree}}(\mathbf{ltree}) = \frac{\partial}{\partial\,\mathbf{ltree}}(a + \mathbf{list}(\mathbf{ltree}))$?
- A hack: $$\begin{aligned}\frac{\partial}{\partial\,\mathbf{ltree}} &= ((\mathbf{1})) + \frac{\partial}{\partial\,\mathbf{ltree}}(a + \mathbf{list}(\mathbf{ltree})) \\ &= \mathbf{1} + \frac{\partial}{\partial\,\mathbf{ltree}}(\mathbf{list}(\mathbf{ltree})) \\ &= \mathbf{1} + \mathbf{list}^2(\mathbf{ltree}) \times \frac{\partial}{\partial\,\mathbf{ltree}}(\mathbf{ltree}) \\ &= \mathbf{ltree\_context}\end{aligned}$$

- $\frac{\partial}{\partial a}\,\mathbf{tree}(a) = \mathbf{list}(\mathbf{tree}(a)) \times \mathbf{list}(a \times \mathbf{list}^2(\mathbf{tree}(a)))$
- Reversed interpretation of the recursion path:





Subtraction and Division?

- $\textbf{list}(a) = 1 + a \times \textbf{list}(a)$

Subtraction and Division?

- $\textbf{list}(a) = 1 + a \times \textbf{list}(a)$
- $\textbf{list}(a) \leftrightarrow \frac{1}{1-a}$?

Subtraction and Division?

- $\textbf{list}(a) = 1 + a \times \textbf{list}(a)$
- $\textbf{list}(a) \leftrightarrow \frac{1}{1-a}$?
- $\frac{\partial}{\partial \mathrm{a}} \textbf{list}(a) \leftrightarrow \frac{\partial}{\partial \mathrm{a}}(\frac{1}{1-a}) = \frac{1}{(1-a)^2} \leftrightarrow \textbf{list}^2(a)$?!

Outline

Species

- General definition: endofunctor on the category of finite sets

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of “labels” to a set of its “arrangements”
- Examples: $\{1, 2, 3\} \rightarrow$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of “labels” to a set of its “arrangements”
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \dots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \dots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$
  - species of sets: $\big\{\{1, 2, 3\}\big\}$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \dots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$
  - species of sets: $\big\{\{1, 2, 3\}\big\}$
  - species of pairs: $\{\}$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$
  - species of sets: $\big\{\{1, 2, 3\}\big\}$
  - species of pairs: $\{\}$
  - species of triplets: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \rightarrow$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$
  - species of sets: $\big\{\{1, 2, 3\}\big\}$
  - species of pairs: $\{\}$
  - species of triplets: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$
  - ...

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \dots\}$
  - species of partitions: $\Big\{\{\{1\}, \{2\}, \{3\}\}, \{\{1\}, \{2, 3\}\}, \cdots\Big\}$
  - species of sets: $\{\{1, 2, 3\}\}$
  - species of pairs: $\{\}$
  - species of triplets: $\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \dots\}$
  - ...
- Species, as functors, preserve identity arrow and composition of arrows

Species

- General definition: endofunctor on the category of finite sets
- Map every finite set of "labels" to a set of its "arrangements"
- Examples: $\{1, 2, 3\} \to$
  - species of permutations: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$
  - species of partitions: $\Big\{\big\{\{1\}, \{2\}, \{3\}\big\}, \big\{\{1\}, \{2, 3\}\big\}, \cdots\Big\}$
  - species of sets: $\big\{\{1, 2, 3\}\big\}$
  - species of pairs: $\{\}$
  - species of triplets: $\big\{(1, 2, 3), (1, 3, 2), (2, 3, 1), \ldots\big\}$
  - ...
- Species, as functors, preserve identity arrow and composition of arrows
  - should be obvious for *regular* species

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\ldots\} \rightarrow \{\}$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\dots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\ldots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\ldots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$
- $F + G$: $S \to F(S) \sqcup G(S)$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\dots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$
- $F + G$: $S \to F(S) \sqcup G(S)$
- $F \cdot G$: $S \to \bigcup_{S_1 \oplus S_2 = L} (F(S_1) \times G(S_2))$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\dots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$
- $F + G$: $S \to F(S) \sqcup G(S)$
- $F \cdot G$: $S \to \bigcup_{S_1 \oplus S_2 = L} (F(S_1) \times G(S_2))$
- $n \triangleq \underbrace{1 + (1 + (\dots + 1))}_{n}$: $\{\} \to \{\{\}_1, \{\}_2, \dots, \{\}_n\}$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\dots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$
- $F + G$: $S \to F(S) \sqcup G(S)$
- $F \cdot G$: $S \to \bigcup_{S_1 \oplus S_2 = L} (F(S_1) \times G(S_2))$
- $n \triangleq \underbrace{1 + (1 + (\dots + 1))}_{n}$: $\{\} \to \{\{\}_1, \{\}_2, \dots, \{\}_n\}$
- $X^n \triangleq \underbrace{X \cdot (X \cdot (\dots \cdot X))}_{n}$: $\{1, 2, \dots, n\} \to \{\text{permutations}\}$

Regular Species

- Composition of 0, 1, X, $+$, $\cdot$ and least fix-point
- 0: $\{\dots\} \to \{\}$
- 1: $\{\} \to \{\{\}\}$, otherwise $\{\}$
- X: $\{x\} \to \{\{x\}\}$, otherwise $\{\}$
- $F + G$: $S \to F(S) \sqcup G(S)$
- $F \cdot G$: $S \to \bigcup\limits_{S_1 \oplus S_2 = L} (F(S_1) \times G(S_2))$
- $n \triangleq \underbrace{1 + (1 + (\dots + 1))}_{n}$: $\{\} \to \{\{\}_1, \{\}_2, \dots, \{\}_n\}$
- $X^n \triangleq \underbrace{X \cdot (X \cdot (\dots \cdot X))}_{n}$: $\{1, 2, \dots, n\} \to \{\text{permutations}\}$
- $L \triangleq 1 + X \cdot L = 1 + X \cdot (1 + X \cdot (\dots)) \simeq 1 + X + X^2 + \dots$

Exponential Generating Function

- Counting function: $C_F : |L| \to |F(L)|$ for species F
- Exponetial Generating Function:
  $E_F(x) = C_F(0) + C_F(1) * \frac{x}{1!} + C_F(2) * \frac{x^2}{2!} + \dots$
- $E_0(x) = 0$
- $E_1(x) = 1$
- $$\begin{aligned} E_{F+G}(x) &= (C_F(0) + C_G(0)) + (C_f(1) + C_G(1)) * \frac{x}{1!} + \dots \\ &= E_F(x) + E_G(x) \end{aligned}$$
- $$\begin{aligned} C_{F \cdot G}(i) * \frac{x^i}{i!} &= \sum_{j=0}^{i} \frac{x^i}{i!} * \binom{i}{j} * C_F(j) * C_G(i-j) \\ &= \sum_{j=0}^{i} \frac{x^j}{j!} * C_F(j) * \frac{x^{i-j}}{(i-j)!} * C_G(i-j) \end{aligned}$$
- $E_{F \cdot G}(x) = E_F(x) * E_G(x)$
- E_F and F share the same expression!

Derivative of Regular Species

- $F' : S \to F(S \cup \{\Box\})$
  - e.g. $(X^2)' : \{1\} \to X^2(\{1, \Box\}) = \{(1, \Box), (\Box, 1)\}$
- Derivative rules apply:
  - $(F + G)' = F' + G'$
  - $(F \cdot G)' = F' \cdot G + F \cdot G'$
  - $\ldots$
- $(X^n)' = n * X^{n-1}$
- $E_{(X^n)'}(x) = E_{n \cdot X^{n-1}}(x) = n * x^{n-1} = (x^n)' = (E_{X^n}(x))'$
- $E_{F'} = (E_F)'$
- Derivative preserves the consistency between regular species and its counting function!

List, Revisit

- $L = 1 + X \cdot L$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$
- $E_L(x) = \frac{1}{1-x}$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$
- $E_L(x) = \frac{1}{1-x}$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$
- $E_L(x) = \frac{1}{1-x}$
- $L = \frac{1}{1-X} =?$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$
- $E_L(x) = \frac{1}{1-x}$
- $L = \frac{1}{1-X} = (\textbf{THE}\ F.\ E_F(x) = \frac{1}{1-x})$

List, Revisit

- $L = 1 + X \cdot L$
- $E_L(x) = 1 + x \cdot E_L(x)$
- $E_L(x) = \frac{1}{1-x}$
- $L = \frac{1}{1-X} = (\textbf{THE } F.\ E_F(x) = \frac{1}{1-x})$
  - $E_F = E_G \Longrightarrow F \simeq G$

Outline

Overview

Zipper Examples

Context as a Derivative

Combinatorial Species

Conclusion

Conclusion

- "Functional pointer": context type
- The structure of context of ordered tree resembles a zipper
- Differentiating an algebraic datatype
- Combinatorial species and its EGF
- Regular species vs. algebraic datatype?

Thank you for listening!

Reference

1. Huet, Gérard. "The zipper." Journal of functional programming 7.5 (1997): 549-554.
2. McBride, Conor. "The derivative of a regular type is its type of one-hole contexts." Unpublished manuscript (2001): 74-88.
3. Yorgey, Brent. "Functional Pearl: Species and Functors and Types, Oh My!."