Random Access Zipper

RAZ

Presented by Kyle Headley
Persistent data structures offer various trade-offs for programmers.

What do we have for sequences?
Zippers are great
Zippers are great

```
type 'a list =  
  | Nil  
  | Cons of 'a * 'a list  

type 'a zip =  
  'a list * 'a list
```
Zippers are great

- Type `a list =` | Nil | Cons of `a * a list`
- Type `a zip =` | `a list * a list`

- Move: `dir -> a zip -> a zip`
- Insert: `dir -> a -> a zip -> a zip`
- Remove: `dir -> a zip -> a zip`

All O(1)!
Zippers are great

Problem: Slow random access

type 'a list =
  | Nil
  | Cons of 'a * 'a list

type 'a zip =
  'a list * 'a list

move: dir ->
  'a zip -> 'a zip

insert: dir -> 'a ->
  'a zip -> 'a zip

remove: dir ->
  'a zip -> 'a zip
Zippers are great

Problem: Slow random access

type 'a list =
| Nil
| Cons of 'a * 'a list

type 'a zip =
'a list * 'a list

move: dir ->
'a zip -> 'a zip
insert: dir -> 'a ->
'a zip -> 'a zip
remove: dir ->
'a zip -> 'a zip

Requires n moves
Trees are great
Trees are great

type 'a tree =
| Nil
| Leaf of 'a
| Bin of 'a tree * 'a tree
Trees are great

type 'a tree =
 | Nil
 | Leaf of 'a
 | Bin of 'a tree * 'a tree

insert: pos -> 'a -> 'a tree -> 'a tree
find: pos -> 'a tree -> 'a
remove: pos -> 'a tree -> 'a tree

All O(log n)! (w/meta data)
Trees are great

```ocaml
type 'a tree =
| Nil
| Leaf of 'a
| Bin of 'a tree * 'a tree
```

Problem: Reasoning about edits

```ocaml
insert: pos -> 'a -> 'a tree ->
     'a tree
find:   pos -> 'a tree -> 'a
remove: pos ->
     'a tree -> 'a tree
```
Trees are great

```ocaml
type 'a tree =
| Nil
| Leaf of 'a
| Bin of 'a tree * 'a tree
```

Problem: Reasoning about edits

- `insert`: `pos -> 'a -> 'a tree -> 'a tree`
- `find`: `pos -> 'a tree -> 'a`
- `remove`: `pos -> 'a tree -> 'a tree`

```
insert here?
```

```
How does rebalance work?
```
Fingertrees are great
Fingertrees are great

\[
\text{first: } 'a \text{ finger} \rightarrow 'a \\
\text{last: } 'a \text{ finger} \rightarrow 'a \\
\text{cons: } 'a \rightarrow \\
\quad 'a \text{ finger} \rightarrow 'a \text{ finger} \\
\text{snoc: } 'a \rightarrow \\
\quad 'a \text{ finger} \rightarrow 'a \text{ finger}
\]

All O(1)! (amortized)
Fingertrees are great

- **first**: \('a\) finger -> \('a\)
- **last**: \('a\) finger -> \('a\)
- **cons**: \('a\) ->
  - \('a\) finger -> \('a\) finger
- **snoc**: \('a\) ->
  - \('a\) finger -> \('a\) finger
- **split**: pos -> \('a\) finger ->
  - \('a\) finger, \('a\) finger
- **append**: \('a\) finger -> \('a\) finger
  - \('a\) finger -> \('a\) finger

Both \(O(\log n)\)!
Fingertrees are great

Problem: Not so simple

first: ‘a finger -> ‘a
last: ‘a finger -> ‘a
cons: ‘a ->
    ‘a finger -> ‘a finger
snoc: ‘a ->
    ‘a finger -> ‘a finger

split: pos -> ‘a finger ->
    (‘a finger, ‘a finger)
append: ‘a finger -> ‘a finger
    -> ‘a finger -> ‘a finger
Fingertrees are great

Problem: Not so simple

```ocaml
type 'a node =
  | Node2 of 'a * 'a
  | Node3 of 'a * 'a * 'a

type 'a digit =
  | One of 'a
  | Two of 'a * 'a
  | Three of 'a * 'a * 'a
  | Four of 'a * 'a * 'a * 'a

type 'a finger =
  | Nil
  | Single of 'a
  | Deep of
    | 'a digit
    * ('a node) finger
    * 'a digit

first:  'a finger -> 'a
last:   'a finger -> 'a
cons:   'a ->
  'a finger -> 'a finger
snoc:   'a ->
  'a finger -> 'a finger

split:  pos -> 'a finger ->
  ('a finger, 'a finger)
append: 'a finger ->
  'a finger -> 'a finger
```
Alternative:
Random Access Zipper

- Accessible
-Editable
- Simple
Using a RAZ

d e

r a z   a b c d e
Using a RAZ

raz

Focused Element
Using a RAZ

|> insert left n   a b c d e
   a n b c d e
Using a RAZ

raz
|> insert left n
|> remove left
Using a RAZ

<table>
<thead>
<tr>
<th>a b c d e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a n b c d e</td>
</tr>
<tr>
<td>a b c d e</td>
</tr>
<tr>
<td>a b d e</td>
</tr>
</tbody>
</table>
Using a RAZ

|> insert left n  a b c d e
|> remove left    a n b c d e
|> remove right  a b c d e
|> unfocus       a b d e
Using a RAZ

<table>
<thead>
<tr>
<th>raz</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt; insert left n</td>
</tr>
<tr>
<td>&gt;&gt; remove left</td>
</tr>
<tr>
<td>&gt;&gt; remove right</td>
</tr>
<tr>
<td>&gt;&gt; unfocus</td>
</tr>
<tr>
<td>&gt;&gt; focus 0</td>
</tr>
</tbody>
</table>

Refocus for random access
Using a RAZ

<table>
<thead>
<tr>
<th>raz</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; insert left n</td>
<td>a</td>
<td>n</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>-&gt; remove left</td>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>-&gt; remove right</td>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>-&gt; unfocus</td>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>-&gt; focus 0</td>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>-&gt; alter right n</td>
<td></td>
<td>a</td>
<td>n</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>
The RAZ is great
The RAZ is great

type 'a tree =
| Nil
| Leaf of 'a
| Bin of lev * item_c
  * 'a tree * 'a tree

A Tree
The RAZ is great

type 'a tree =
| Nil
| Leaf of 'a
| Bin of lev * item_c
  * 'a tree * 'a tree

type 'a list =
| Nil
| Cons of 'a * 'a list
| Level of lev * 'a list
| Tree of 'a tree * 'a list
The RAZ is great

type 'a tree =
  | Nil
  | Leaf of 'a
  | Bin of \texttt{lev} * \texttt{item_c}
    * 'a tree * 'a tree

type 'a list =
  | Nil
  | Cons of 'a * 'a list
  | Level of lev * 'a list
  | Tree of 'a tree * 'a list

type 'a raz =
  'a list * 'a * 'a list

As a zipper
The RAZ is great

```ocaml
type 'a tree =
 | Nil
 | Leaf of 'a
 | Bin of lev * item_c
    * 'a tree * 'a tree

type 'a list =
 | Nil
 | Cons of 'a * 'a list
 | Level of lev * 'a list
 | Tree of 'a tree * 'a list

type 'a raz =
 'a list * 'a * 'a list
```
The RAZ is great

```ocaml
type 'a tree =
| Nil
| Leaf of 'a
| Bin of lev * item_c
     * 'a tree * 'a tree

type 'a list =
| Nil
| Cons of 'a * 'a list
| Level of lev * 'a list
| Tree of 'a tree * 'a list

type 'a raz =
  'a list * 'a * 'a list

move: dir ->
     'a zip -> 'a zip
insert: dir -> 'a ->
     'a zip -> 'a zip
remove: dir ->
     'a zip -> 'a zip

All O(1)!
```
The RAZ is great

**type 'a tree =**
| Nil
| Leaf of 'a
| Bin of lev * item_c
  * 'a tree * 'a tree

**type 'a list =**
| Nil
| Cons of 'a * 'a list
| Level of lev * 'a list
| Tree of 'a tree * 'a list

**type 'a raz =**
 'a list * 'a * 'a list

**move: dir ->**
 'a zip -> 'a zip

**insert: dir -> 'a ->**
 'a zip -> 'a zip

**remove: dir ->**
 'a zip -> 'a zip

**focus: val ->**
 'a tree -> 'a raz

**unfocus: 'a raz -> 'a tree**

Both O(log n)! (plus net insertions)
The RAZ is great

Simple: < 200 lines of code
Zipper of Trees
Zipper of Trees

Focus

Nil → a → b → 2 → 9 → 6 → c → d → e → Nil
Zipper of Trees

List element with data

Focus

List element with tree

Levels

Leaves with data

Tree Nodes w/ Levels
Levels for Balance
Levels provide a way to maintain the balance of the tree elements of a RAZ
Levels provide a way to maintain the balance of the tree elements of a RAZ.

Random levels drawn from distribution of levels in a (huge) binary tree.
Levels provide a way to maintain the balance of the tree elements of a RAZ.

Random levels drawn from distribution of levels in a (huge) binary tree.
Levels provide a way to maintain the balance of the tree elements of a RAZ

Random levels drawn from distribution of levels in a (huge) binary tree
Balance

Because of the way randomness behaves, we get good balance at scale.
Balance

Because of the way randomness behaves, we get good balance at scale.

Levels track tree balance, and we store them in list nodes so that height is not lost when deconstructed.
Two Forms of RAZ
Two Forms of RAZ

Zipper

a \rightarrow b \rightarrow c \rightarrow d \rightarrow e
Two Forms of RAZ

Editable: make insertions and deletions just like a common linked list

Zipper

a → b → c → d → e

2 ← 9 → 1 → 6
Two Forms of RAZ

Tree

a → b → 2 → 9 → c → 1 → d → 6 → e
Two Forms of RAZ

Tree

A single balanced binary tree: efficient searching algorithms
Two Forms of RAZ

Tree

Zipper
Two Forms of RAZ

Tree

Zipper
Two Forms of RAZ

Tree

Zipper

Unfocus
Focusing
Focusing
Focusing
Focusing
Focusing
Focusing
Focusing
Focusing
Experiments
Experiments

RAZ in OCaml
Experiments

RAZ in OCaml

Fingertree in OCaml
Experiments

RAZ in OCaml

Fingertree in OCaml

Build a sequence by insertions at random points
Insertion at random

Single Insertions

Time (s)

Insertions (x100k)

RAZ
Fingertree
Random Access Zipper

- Accessible
- Editable
- Simple
- Fast

Focus/Unfocus
No edit rebalance
< 200 LoC
Beats Fingertree
Random Access Zipper

Simple enough to include these principles in your own data types!

More info at kyleheadley.github.io