Random Access List

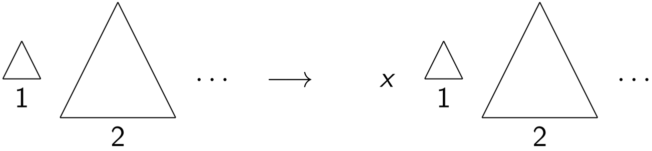
Chris Okasaki

- Interface:
  - `cons:  T -> ralist -> ralist` $O(1)$
  - `head:  ralist -> option T` $O(1)$
  - `tail:  ralist -> ralist` $O(1)$
  - `get :  ralist -> nat -> option T` $O(\log n)$
  - `set :  ralist -> nat -> T -> ralist` $O(\log n)$
- Representation:
  - List of balanced trees with nodes labeled by elements of `T`.
  - Trees of the list are of strictly increasing height.
    Exception: the first two trees may have the same height.
  - The older the elements, the farther in the list of trees they are.
    Elements in a tree are stored with a depth-first pre-order traversal.

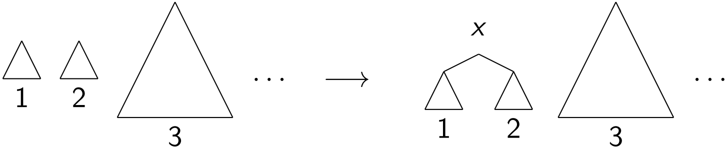
Random Access List

Adding an element to a list

- If the first two trees have different heights,



- If the first two trees have the same height,



Coq Types

```
Variable T : Type.

Inductive tree :=
  | Leaf : T -> tree
  | Node : T -> tree -> tree -> tree.

Inductive ralist :=
  | raNil : ralist
  | raCons : tree -> nat -> ralist -> ralist.
```

Definition of Head

```
Definition head l :=
  match l with
  | raNil => None
  | raCons t _ _ =>
    match t with
    | Leaf x => Some x
    | Node x _ _ => Some x
    end
  end.
```

Definition of Cons

```
Definition cons x l :=
  match l with
  | raNil => raCons (Leaf x) 0 l
  | raCons t s raNil => raCons (Leaf x) 0 l
  | raCons t1 h1 (raCons t2 h2 q) =>
    if h1 == h2 then raCons (Node x t1 t2) (1 + h1) q
    else raCons (Leaf x) 0 l
  end.
```

Definition of Tail

```
Definition tail l :=
  match l with
  | raNil => raNil
  | raCons t h q =>
    match t with
    | Leaf _ => q
    | Node _ t1 t2 =>
      raCons t1 (h - 1) (raCons t2 (h - 1) q)
    end
  end.
```

RA Lists are Lists

```
Lemma head_cons :
  forall l x,
  head (cons x l) = Some x.

Lemma tail_cons :
  forall l x,
  tail (cons x l) = l.
```

Data Invariant

```
Fixpoint height t :=
  match t with
  | Leaf _ => 0
  | Node _ t1 _ => 1 + height t1
  end.

Fixpoint balanced t :=
  match t with
  | Leaf _ => True
  | Node _ t1 t2 =>
    height t1 = height t2 /\ balanced t1 /\ balanced
        t2
  end.
```

Data Invariant

```
Fixpoint structured_aux l h :=
  match l with
  | raNil => True
  | raCons t h' q =>
    balanced t /\ height t = h' /\ h <= h' /\
    structured_aux q (1 + h')
  end.

Definition structured l :=
  match l with
  | raNil => True
  | raCons t h q =>
    balanced t /\ height t = h /\
    structured_aux q h
  end.
```

Preservation of Invariant

```
Lemma structured_cons :
  forall l x,
  structured l ->
  structured (cons x l).

Lemma structured_tail :
  forall l,
  structured l ->
  structured (tail l).
```