

Treaps

Max Haslbeck, Manuel Eberl, Tobias Nipkow

May 21, 2018

Abstract

A Treap [2] is a binary tree whose nodes contain pairs consisting of some payload and an associated priority. It must have the search-tree property w.r.t. the payloads and the heap property w.r.t. the priorities. Treaps are an interesting data structure that is related to binary search trees (BSTs) in the following way: if one forgets all the priorities of a treap, the resulting BST is exactly the same as if one had inserted the elements into an empty BST in order of ascending priority. This means that a treap behaves like a BST where we can pretend the elements were inserted in a different order from the one in which they were actually inserted.

In particular, by choosing these priorities at random upon insertion of an element, we can pretend that we inserted the elements in *random order*, so that the shape of the resulting tree is that of a random BST no matter in what order we insert the elements. This is the main result of this formalisation. [1]

Contents

1	Auxiliary material	2
2	Treaps	4
3	Randomly-permuted lists	7
3.1	General facts about linear orderings	7
4	Relationship between treaps and BSTs	11
5	Random treaps	12
5.1	Measurability	12
5.2	Main result	15

1 Auxiliary material

theory *Probability-Misc*

imports *HOL-Probability.Probability*

begin

lemma *measure-eqI-countable-AE'*:

assumes [*simp*]: *sets M = Pow B sets N = Pow B and subset: $\Omega \subseteq B$*

assumes *ae: AE x in M. x \in Ω AE x in N. x \in Ω and [simp]: countable Ω*

assumes *eq: $\bigwedge x. x \in \Omega \implies \text{emeasure } M \{x\} = \text{emeasure } N \{x\}$*

shows *M = N*

<proof>

lemma *measurable-le[measurable (raw)]*:

fixes *f :: 'a \Rightarrow 'b::{second-countable-topology, linorder-topology}*

assumes *f \in borel-measurable M g \in borel-measurable M*

shows *Measurable.pred M ($\lambda x. f x \leq g x$)*

<proof>

lemma *measurable-eq[measurable (raw)]*:

fixes *f :: 'a \Rightarrow 'b::{second-countable-topology, linorder-topology}*

assumes *f \in borel-measurable M g \in borel-measurable M*

shows *Measurable.pred M ($\lambda x. f x = g x$)*

<proof>

context

fixes *M :: 'a measure*

assumes *singleton-null-set: x \in space M \implies {x} \in null-sets M*

begin

lemma *countable-null-set*:

assumes *countable A A \subseteq space M*

shows *A \in null-sets M*

<proof>

lemma *finite-null-set*:

assumes *finite A A \subseteq space M*

shows *A \in null-sets M*

<proof>

end

lemma *measurable-inj-on-finite*:

assumes *fin [measurable]: finite I*

assumes [*measurable*]: $\bigwedge i j. \text{Measurable.pred } (M i \otimes_M M j) (\lambda(x,y). x = y)$

shows *Measurable.pred (Pi_M I M) ($\lambda x. \text{inj-on } x I$)* *<proof>*

lemma *almost-everywhere-not-in-countable-set*:

assumes *countable A*

assumes *[measurable]*: *Measurable.pred* ($M \otimes_M M$) ($\lambda(x,y). x = y$)
assumes *null*: $\bigwedge x. x \in \text{space } M \implies \{x\} \in \text{null-sets } M$
shows *AE* x in $M. x \notin A$
 <proof>

lemma *almost-everywhere-inj-on-PiM*:
assumes *fin*: *finite* I **and** *prob-space*: $\bigwedge i. i \in I \implies \text{prob-space } (M i)$
assumes *[measurable]*: $\bigwedge i j. \text{Measurable.pred } (M i \otimes_M M j) (\lambda(x,y). x = y)$
assumes *null*: $\bigwedge i x. i \in I \implies x \in \text{space } (M i) \implies \{x\} \in \text{null-sets } (M i)$
shows *AE* f in $(\prod_M i \in I. M i). \text{inj-on } f I$
 <proof>

lemma *null-sets-uniform-measure*:
assumes $A \in \text{sets } M$ *emeasure* $M A \neq \infty$
shows $\text{null-sets } (\text{uniform-measure } M A) = (\lambda B. A \cap B) - \text{null-sets } M \cap \text{sets } M$
 <proof>

lemma *almost-everywhere-avoid-finite*:
assumes *fin*: *finite* I
shows *AE* f in $(\prod_M i \in I. \text{uniform-measure lborel } \{(0::\text{real})..1\}). \text{inj-on } f I$
 <proof>

lemma *almost-everywhere-avoid-countable*:
assumes *countable* A
shows *AE* x in $\text{uniform-measure lborel } \{(0::\text{real})..1\}. x \notin A$
 <proof>

lemma *measure-pmf-of-set*:
assumes $A \neq \{\}$ **and** *finite* A
shows $\text{measure-pmf } (\text{pmf-of-set } A) = \text{uniform-measure } (\text{count-space UNIV}) A$
 <proof>

lemma *emeasure-distr-restrict*:
assumes $f \in M \rightarrow_M N$ $f \in M' \rightarrow_M N'$ $A \in \text{sets } N'$ $\text{sets } M' \subseteq \text{sets } M$ $\text{sets } N' \subseteq \text{sets } N$
assumes $\bigwedge X. X \in \text{sets } M' \implies \text{emeasure } M X = \text{emeasure } M' X$
assumes $\bigwedge X. X \in \text{sets } M \implies X \subseteq \text{space } M - \text{space } M' \implies \text{emeasure } M X = 0$
shows $\text{emeasure } (\text{distr } M N f) A = \text{emeasure } (\text{distr } M' N' f) A$
 <proof>

lemma *distr-uniform-measure-count-space-inj*:
assumes *inj-on* $f A' A' \subseteq A$ $f ' A \subseteq B$ *finite* A'
shows $\text{distr } (\text{uniform-measure } (\text{count-space } A) A') (\text{count-space } B) f = \text{uniform-measure } (\text{count-space } B) (f ' A')$ (**is** *?lhs = ?rhs*)
 <proof>

lemma (in *pair-prob-space*) *pair-measure-bind*:
assumes [*measurable*]: $f \in M1 \otimes_M M2 \rightarrow_M \text{subprob-algebra } N$
shows $(M1 \otimes_M M2) \gg= f = \text{do } \{x \leftarrow M1; y \leftarrow M2; f(x, y)\}$
<proof>

lemma *count-space-singleton-conv-return*:
 $\text{count-space } \{x\} = \text{return } (\text{count-space } \{x\}) x$
<proof>

lemma *distr-count-space-singleton [simp]*:
 $f x \in \text{space } M \implies \text{distr } (\text{count-space } \{x\}) M f = \text{return } M (f x)$
<proof>

lemma *uniform-measure-count-space-singleton [simp]*:
assumes $\{x\} \in \text{sets } M$ *emeasure* $M \{x\} \neq 0$ *emeasure* $M \{x\} < \infty$
shows $\text{uniform-measure } M \{x\} = \text{return } M x$
<proof>

lemma *PiM-uniform-measure-permute*:
fixes $a b :: \text{real}$
assumes g *permutes* $A a < b$
shows $\text{distr } (PiM A (\lambda-. \text{uniform-measure lborel } \{a..b\})) (PiM A (\lambda-. \text{lborel}))$
 $(\lambda f. f \circ g) =$
 $PiM A (\lambda-. \text{uniform-measure lborel } \{a..b\})$
<proof>

lemma *ennreal-fact [simp]*: $\text{ennreal } (\text{fact } n) = \text{fact } n$
<proof>

lemma *inverse-ennreal-unique*:
assumes $a * (b :: \text{ennreal}) = 1$
shows $b = \text{inverse } a$
<proof>

end

2 Treaps

theory *Treap*

imports

HOL-Library.Tree

begin

definition $\text{treap} :: ('k::\text{linorder} * 'p::\text{linorder}) \text{tree} \Rightarrow \text{bool}$ **where**
 $\text{treap } t = (\text{bst } (\text{map-tree fst } t) \wedge \text{heap } (\text{map-tree snd } t))$

abbreviation $\text{keys } t \equiv \text{set-tree } (\text{map-tree fst } t)$

abbreviation $\text{prios } t \equiv \text{set-tree } (\text{map-tree snd } t)$

function *treap-of* :: ('k::linorder * 'p::linorder) set \Rightarrow ('k * 'p) tree **where**
treap-of KP = (if infinite KP \vee KP = {} then Leaf else
 let m = arg-min-on snd KP;
 L = {p \in KP. fst p < fst m};
 R = {p \in KP. fst p > fst m}
 in Node (treap-of L) m (treap-of R))
 <proof>
termination
 <proof>

declare *treap-of.simps* [simp del]

lemma *treap-of-unique*:
 [[treap t; inj-on snd (set-tree t)]]
 \implies treap-of (set-tree t) = t
 <proof>

lemma *treap-unique*:
 [[treap t1; treap t2; set-tree t1 = set-tree t2; inj-on snd (set-tree t1)]]
 \implies t1 = t2
for t1 t2 :: ('k::linorder * 'p::linorder) tree
 <proof>

fun *ins* :: 'k::linorder \Rightarrow 'p::linorder \Rightarrow ('k \times 'p) tree \Rightarrow ('k \times 'p) tree **where**
ins k p Leaf = <Leaf, (k,p), Leaf> |
ins k p <l, (k1,p1), r> =
 (if k < k1 then
 (case *ins* k p l of
 <l2, (k2,p2), r2> \Rightarrow
 if p1 \leq p2 then <<l2, (k2,p2), r2>, (k1,p1), r>
 else <l2, (k2,p2), <r2, (k1,p1), r>>
 else
 if k > k1 then
 (case *ins* k p r of
 <l2, (k2,p2), r2> \Rightarrow
 if p1 \leq p2 then <l, (k1,p1), <l2, (k2,p2), r2>>
 else <<l, (k1,p1), l2>, (k2,p2), r2>>
 else <l, (k1,p1), r>))

lemma *ins-neq-Leaf*: *ins* k p t \neq <>
 <proof>

lemma *keys-ins*: keys (*ins* k p t) = Set.insert k (keys t)
 <proof>

lemma *prios-ins*: prios (*ins* k p t) \subseteq {p} \cup prios t
 <proof>

lemma *prios-ins'*: k \notin keys t \implies prios (*ins* k p t) = {p} \cup prios t

$\langle \text{proof} \rangle$

lemma *set-tree-ins*: $\text{set-tree } (\text{ins } k \ p \ t) \subseteq \{(k,p)\} \cup \text{set-tree } t$
 $\langle \text{proof} \rangle$

lemma *set-tree-ins'*: $k \notin \text{keys } t \implies \{(k,p)\} \cup \text{set-tree } t \subseteq \text{set-tree } (\text{ins } k \ p \ t)$
 $\langle \text{proof} \rangle$

lemma *set-tree-ins-eq*: $k \notin \text{keys } t \implies \text{set-tree } (\text{ins } k \ p \ t) = \{(k,p)\} \cup \text{set-tree } t$
 $\langle \text{proof} \rangle$

lemma *prios-ins-special*:
[[$\text{ins } k \ p \ t = \text{Node } l \ (k',p') \ r$; $p' = p$; $p \in \text{prios } r \cup \text{prios } l$]]
 $\implies p \in \text{prios } t$
 $\langle \text{proof} \rangle$

lemma *treap-NodeI*:
[[$\text{treap } l$; $\text{treap } r$;
 $\forall k' \in \text{keys } l. k' < k$; $\forall k' \in \text{keys } r. k < k'$;
 $\forall p' \in \text{prios } l. p \leq p'$; $\forall p' \in \text{prios } r. p \leq p'$]]
 $\implies \text{treap } (\text{Node } l \ (k,p) \ r)$
 $\langle \text{proof} \rangle$

lemma *treap-rotate1*:
assumes $\text{treap } l2 \ \text{treap } r2 \ \text{treap } r \ \neg p1 \leq p2 \ k < k1$ **and**
ins: $\text{ins } k \ p \ l = \text{Node } l2 \ (k2,p2) \ r2$ **and** *treap-ins*: $\text{treap } (\text{ins } k \ p \ l)$
and *treap*: $\text{treap } \langle l, (k1, p1), r \rangle$
shows $\text{treap } (\text{Node } l2 \ (k2,p2) \ (\text{Node } r2 \ (k1,p1) \ r))$
 $\langle \text{proof} \rangle$

lemma *treap-rotate2*:
assumes $\text{treap } l \ \text{treap } l2 \ \text{treap } r2 \ \neg p1 \leq p2 \ k1 < k$ **and**
ins: $\text{ins } k \ p \ r = \text{Node } l2 \ (k2,p2) \ r2$ **and** *treap-ins*: $\text{treap } (\text{ins } k \ p \ r)$
and *treap*: $\text{treap } \langle l, (k1, p1), r \rangle$
shows $\text{treap } (\text{Node } (\text{Node } l \ (k1,p1) \ l2) \ (k2,p2) \ r2)$
 $\langle \text{proof} \rangle$

lemma *treap-ins*: $\text{treap } t \implies \text{treap } (\text{ins } k \ p \ t)$
 $\langle \text{proof} \rangle$

lemma *treap-of-set-tree-unique*:
[[$\text{finite } A$; $\text{inj-on fst } A$; $\text{inj-on snd } A$]]
 $\implies \text{set-tree } (\text{treap-of } A) = A$
 $\langle \text{proof} \rangle$

lemma *treap-of-subset*: $\text{set-tree } (\text{treap-of } A) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *treap-treap-of*:

treap (treap-of A)
<proof>

lemma *treap-Leaf*: *treap <>*

<proof>

lemma *foldl-ins-treap*: *treap t \implies treap (foldl ($\lambda t' (x, p). ins x p t'$) t xs)*

<proof>

lemma *foldl-ins-set-tree*:

assumes *inj-on fst (set ys) inj-on snd (set ys) distinct ys fst ' (set ys) \cap keys t*
= {}

shows *set-tree (foldl ($\lambda t' (x, p). ins x p t'$) t ys) = set ys \cup set-tree t*

<proof>

lemma *foldl-ins-treap-of*:

assumes *distinct ys inj-on fst (set ys) inj-on snd (set ys)*

shows *(foldl ($\lambda t' (x, p). ins x p t'$) Leaf ys) = treap-of (set ys)*

<proof>

end

3 Randomly-permuted lists

theory *Random-List-Permutation*

imports

Probability-Misc

Comparison-Sort-Lower-Bound.Linorder-Relations

begin

3.1 General facts about linear orderings

We define the set of all linear orderings on a given set and show some properties about it.

definition *linorders-on* :: *'a set \Rightarrow ('a \times 'a) set set* **where**

linorders-on A = {R. linorder-on A R}

lemma *linorders-on-empty* [*simp*]: *linorders-on {} = {{}}*

<proof>

lemma *linorders-finite-nonempty*:

assumes *finite A*

shows *linorders-on A \neq {}*

<proof>

There is an obvious bijection between permutations of a set (i. e. lists with all elements from that set without repetition) and linear orderings on it.

lemma *bij-betw-linorders-on*:

assumes *finite A*

shows *bij-betw linorder-of-list (permutations-of-set A) (linorders-on A)*

<proof>

lemma *sorted-wrt-list-of-set-linorder-of-list [simp]*:

assumes *distinct xs*

shows *sorted-wrt-list-of-set (linorder-of-list xs) (set xs) = xs*

<proof>

lemma *linorder-of-list-sorted-wrt-list-of-set [simp]*:

assumes *linorder-on A R finite A*

shows *linorder-of-list (sorted-wrt-list-of-set R A) = R*

<proof>

lemma *bij-betw-linorders-on'*:

assumes *finite A*

shows *bij-betw ($\lambda R. \text{sorted-wrt-list-of-set } R \ A$) (linorders-on A) (permutations-of-set A)*

<proof>

lemma *finite-linorders-on [intro]*:

assumes *finite A*

shows *finite (linorders-on A)*

<proof>

Next, we look at the ordering defined by a list that is permuted with some permutation function. For this, we first define the composition of a relation with a function.

definition *map-relation* :: *'a set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \times 'b) set \Rightarrow ('a \times 'a) set*
where

map-relation A f R = {(x,y) \in A \times A. (f x, f y) \in R}

lemma *index-distinct-eqI*:

assumes *distinct xs i < length xs xs ! i = x*

shows *index xs x = i*

<proof>

lemma *index-permute-list*:

assumes *π permutes {.. $\text{length } xs$ } distinct xs $x \in \text{set } xs$*

shows *index (permute-list π xs) x = inv π (index xs x)*

<proof>

lemma *linorder-of-list-permute*:

assumes *π permutes {.. $\text{length } xs$ } distinct xs*

shows *linorder-of-list (permute-list π xs) =*

map-relation (set xs) (!) xs \circ inv π \circ index xs (linorder-of-list xs)

<proof>

lemma *inj-on-conv-Ex1*: $\text{inj-on } f \ A \longleftrightarrow (\forall y \in f' A. \exists ! x \in A. y = f x)$
 ⟨proof⟩

lemma *bij-betw-conv-Ex1*: $\text{bij-betw } f \ A \ B \longleftrightarrow (\forall y \in B. \exists ! x \in A. f x = y) \wedge B = f ' A$
 ⟨proof⟩

lemma *permutesI*:
assumes $\text{bij-betw } f \ A \ A \ \forall x. x \notin A \longrightarrow f x = x$
shows $f \text{ permutes } A$
 ⟨proof⟩

We now show the important lemma that any two linear orderings on a finite set can be mapped onto each other by a permutation.

lemma *linorder-permutation-exists*:
assumes $\text{finite } A \ \text{linorder-on } A \ R \ \text{linorder-on } A \ R'$
obtains π **where** $\pi \text{ permutes } A \ R' = \text{map-relation } A \ \pi \ R$
 ⟨proof⟩

We now define the linear ordering defined by some priority function, i. e. a function that injectively associates priorities to every element such that elements with lower priority are smaller in the resulting ordering.

definition *linorder-from-keys* :: $'a \ \text{set} \Rightarrow ('a \Rightarrow 'b :: \text{linorder}) \Rightarrow ('a \times 'a) \ \text{set}$
where
 $\text{linorder-from-keys } A \ f = \{(x,y) \in A \times A. f x \leq f y\}$

lemma *linorder-from-keys-permute*:
assumes $g \text{ permutes } A$
shows $\text{linorder-from-keys } A \ (f \circ g) = \text{map-relation } A \ g \ (\text{linorder-from-keys } A \ f)$
 ⟨proof⟩

lemma *linorder-on-linorder-from-keys [intro]*:
assumes $\text{inj-on } f \ A$
shows $\text{linorder-on } A \ (\text{linorder-from-keys } A \ f)$
 ⟨proof⟩

lemma *linorder-from-keys-empty [simp]*: $\text{linorder-from-keys } \{\} = (\lambda \cdot. \{\})$
 ⟨proof⟩

We now show another important fact, namely that when we draw n values i. i. d. uniformly from a non-trivial real interval, we almost surely get distinct values.

lemma *emeasure-PiM-diagonal*:
fixes $a \ b :: \text{real}$
assumes $x \in A \ y \in A \ x \neq y$
assumes $a < b \ \text{finite } A$

defines $M \equiv \text{uniform-measure lborel } \{a..b\}$
shows $\text{emeasure } (PiM A (\lambda-. M)) \{h \in A \rightarrow_E UNIV. h x = h y\} = 0$
 $\langle \text{proof} \rangle$

lemma *measurable-linorder-from-keys-restrict*:

assumes $\text{fin: finite } A$
shows $\text{linorder-from-keys } A \in PiM A (\lambda-. \text{borel} :: \text{real measure}) \rightarrow_M \text{count-space}$
 $(Pow (A \times A))$
(is - : ?M \rightarrow_M -)
 $\langle \text{proof} \rangle$

lemma *measurable-count-space-extend*:

assumes $f \in \text{measurable } M (\text{count-space } A) \ A \subseteq B$
shows $f \in \text{measurable } M (\text{count-space } B)$
 $\langle \text{proof} \rangle$

lemma *measurable-linorder-from-keys-restrict'*:

assumes $\text{fin: finite } A \ A \subseteq B$
shows $\text{linorder-from-keys } A \in PiM A (\lambda-. \text{borel} :: \text{real measure}) \rightarrow_M \text{count-space}$
 $(Pow (B \times B))$
 $\langle \text{proof} \rangle$

context

fixes $a \ b :: \text{real}$ **and** $A :: 'a \text{ set}$ **and** M **and** B
assumes $\text{fin: finite } A$ **and** $\text{ab: } a < b$ **and** $B: A \subseteq B$
defines $M \equiv \text{distr } (PiM A (\lambda-. \text{uniform-measure lborel } \{a..b\}))$
 $(\text{count-space } (Pow (B \times B))) (\text{linorder-from-keys } A)$

begin

lemma *measurable-linorder-from-keys [measurable]*:

$\text{linorder-from-keys } A \in PiM A (\lambda-. \text{borel} :: \text{real measure}) \rightarrow_M \text{count-space } (Pow$
 $(B \times B))$
 $\langle \text{proof} \rangle$

The ordering defined by randomly-chosen priorities is almost surely linear:

theorem *almost-everywhere-linorder*: $AE \ R \text{ in } M. \text{linorder-on } A \ R$

$\langle \text{proof} \rangle$

Furthermore, this is equivalent to choosing one of the $|A|!$ linear orderings uniformly at random.

theorem *random-linorder-by-prios*:

$M = \text{uniform-measure } (\text{count-space } (Pow (B \times B))) (\text{linorders-on } A)$
 $\langle \text{proof} \rangle$

end

end

4 Relationship between treaps and BSTs

```

theory Treap-Sort-and-BSTs
imports
  Treap
  Random-List-Permutation
  Random-BSTs.Random-BSTs
begin

```

Here, we will show that if we “forget” the priorities of a treap, we essentially get a BST into which the elements have been inserted by ascending priority. First, we show some facts about sorting that we will need.

The following two lemmas are only important for measurability later.

lemma *insort-key-conv-rec-list*:

```

insort-key f x xs =
  rec-list [x] ( $\lambda y\ ys\ zs.$  if  $f\ x \leq f\ y$  then  $x \# y \# ys$  else  $y \# zs$ ) xs
<proof>

```

lemma *insort-key-conv-rec-list'*:

```

insort-key = ( $\lambda f\ x.$ 
  rec-list [x] ( $\lambda y\ ys\ zs.$  if  $f\ x \leq f\ y$  then  $x \# y \# ys$  else  $y \# zs$ ))
<proof>

```

lemma *bst-of-list-trees*:

```

assumes set ys  $\subseteq$  A
shows bst-of-list ys  $\in$  trees A
<proof>

```

lemma *insort-wrt-insort-key*:

```

a  $\in$  A  $\implies$ 
set xs  $\subseteq$  A  $\implies$ 
insort-wrt (linorder-from-keys A f) a xs = insort-key f a xs
<proof>

```

lemma *insort-wrt-sort-key*:

```

assumes set xs  $\subseteq$  A
shows insort-wrt (linorder-from-keys A f) xs = sort-key f xs
<proof>

```

The following is an important recurrence for *sort-key* that states that for distinct priorities, sorting a list w. r. t. those priorities can be seen as selection sort, i. e. we can first choose the (unique) element with minimum priority as the first element and then sort the rest of the list and append it.

lemma *sort-key-arg-min-on*:

```

assumes zs  $\neq$  [] inj-on p (set zs)
shows sort-key p (zs::'a::linorder list) =
  (let z = arg-min-on p (set zs) in z # sort-key p (remove1 z zs))
<proof>

```

lemma *arg-min-on-image-finite*:
fixes $f :: 'b \Rightarrow 'c :: \text{linorder}$
assumes $\text{inj-on } f (g \text{ ` } B) \text{ finite } B \ B \neq \{\}$
shows $\text{arg-min-on } f (g \text{ ` } B) = g (\text{arg-min-on } (f \circ g) B)$
 $\langle \text{proof} \rangle$

lemma *fst-snd-arg-min-on*: **fixes** $p :: 'a \Rightarrow 'b :: \text{linorder}$
assumes $\text{finite } B \ \text{inj-on } p \ B \ B \neq \{\}$
shows $\text{fst } (\text{arg-min-on } \text{snd } ((\lambda x. (x, p \ x)) \text{ ` } B)) = \text{arg-min-on } p \ B$
 $\langle \text{proof} \rangle$

The following is now the main result:

theorem *treap-of-bst-of-list'*:
assumes $ys = \text{map } (\lambda x. (x, p \ x)) \ xs \ \text{inj-on } p \ (\text{set } xs) \ xs' = \text{sort-key } p \ xs$
shows $\text{map-tree } \text{fst } (\text{treap-of } (\text{set } ys)) = \text{bst-of-list } xs'$
 $\langle \text{proof} \rangle$

corollary *treap-of-bst-of-list*: $\text{inj-on } p \ (\text{set } zs) \Longrightarrow$
 $\text{map-tree } \text{fst } (\text{treap-of } (\text{set } (\text{map } (\lambda x. (x, p \ x)) \ zs))) = \text{bst-of-list } (\text{sort-key } p \ zs)$
 $\langle \text{proof} \rangle$

corollary *treap-of-bst-of-list''*: $\text{inj-on } p \ (\text{set } zs) \Longrightarrow$
 $\text{map-tree } \text{fst } (\text{treap-of } ((\lambda x. (x, p \ x)) \text{ ` } \text{set } zs)) = \text{bst-of-list } (\text{sort-key } p \ zs)$
 $\langle \text{proof} \rangle$

corollary *fold-ins-bst-of-list*: $\text{distinct } zs \Longrightarrow \text{inj-on } p \ (\text{set } zs) \Longrightarrow$
 $\text{map-tree } \text{fst } (\text{foldl } (\lambda t (x,p). \ \text{ins } x \ p \ t) \ \langle \rangle \ (\text{map } (\lambda x. (x, p \ x)) \ zs)) = \text{bst-of-list}$
 $(\text{sort-key } p \ zs)$
 $\langle \text{proof} \rangle$

end

5 Random treaps

theory *Random-Treap*
imports
Probability-Misc
Treap-Sort-and-BSTs
begin

5.1 Measurability

The following lemmas are only relevant for measurability.

lemma *tree-sigma-cong*:
assumes $\text{sets } M = \text{sets } M'$
shows $\text{tree-sigma } M = \text{tree-sigma } M'$
 $\langle \text{proof} \rangle$

lemma *distr-restrict*:

assumes *sets N = sets L sets K ⊆ sets M*

$\bigwedge X. X \in \text{sets } K \implies \text{emeasure } M X = \text{emeasure } K X$

$\bigwedge X. X \in \text{sets } M \implies X \subseteq \text{space } M - \text{space } K \implies \text{emeasure } M X = 0$

$f \in M \rightarrow_M N \implies f \in K \rightarrow_M L$

shows $\text{distr } M N f = \text{distr } K L f$

<proof>

lemma *sets-tree-sigma-count-space*:

assumes *countable B*

shows $\text{sets } (\text{tree-sigma } (\text{count-space } B)) = \text{Pow } (\text{trees } B)$

<proof>

lemma *height-primrec*: $\text{height} = \text{rec-tree } 0 \ (\lambda - - a b. \text{Suc } (\text{max } a b))$

<proof>

lemma *ipl-primrec*: $\text{ipl} = \text{rec-tree } 0 \ (\lambda l - r a b. \text{size } l + \text{size } r + a + b)$

<proof>

lemma *size-primrec*: $\text{size} = \text{rec-tree } 0 \ (\lambda - - a b. 1 + a + b)$

<proof>

lemma *ipl-map-tree[simp]*: $\text{ipl } (\text{map-tree } f t) = \text{ipl } t$

<proof>

lemma *set-pmf-random-bst*: $\text{finite } A \implies \text{set-pmf } (\text{random-bst } A) \subseteq \text{trees } A$

<proof>

lemma *trees-mono*: $A \subseteq B \implies \text{trees } A \subseteq \text{trees } B$

<proof>

lemma *ins-primrec*:

ins k (p::real) t = rec-tree

(Node Leaf (k,p) Leaf)

(λ l z r l' r'. case z of (k1, p1) ⇒

if k < k1 then

(case l' of

Leaf ⇒ Leaf

| Node l2 (k2,p2) r2 ⇒

if 0 ≤ p2 - p1 then Node (Node l2 (k2,p2) r2) (k1,p1) r

else Node l2 (k2,p2) (Node r2 (k1,p1) r))

else if k > k1 then

(case r' of

Leaf ⇒ Leaf

| Node l2 (k2,p2) r2 ⇒

if 0 ≤ p2 - p1 then Node l (k1,p1) (Node l2 (k2,p2) r2)

else Node (Node l (k1,p1) l2) (k2,p2) r2)

else Node l (k1,p1) r
) t
 ⟨proof⟩

lemma *measurable-less-count-space* [measurable (raw)]:

assumes countable A
assumes [measurable]: a ∈ B →_M count-space A
assumes [measurable]: b ∈ B →_M count-space A
shows Measurable.pred B (λx. a x < b x)
 ⟨proof⟩

lemma *measurable-ins* [measurable (raw)]:

assumes [measurable]: countable A
assumes [measurable]: k ∈ B →_M count-space A
assumes [measurable]: x ∈ B →_M (lborel :: real measure)
assumes [measurable]: t ∈ B →_M tree-sigma (count-space A ⊗_M lborel)
shows (λy. ins (k y) (x y) (t y)) ∈ B →_M tree-sigma (count-space A ⊗_M lborel)
 ⟨proof⟩

lemma *map-tree-primrec*: map-tree f t = rec-tree ⟨⟩ (λl a r l' r'. ⟨l', f a, r'⟩) t
 ⟨proof⟩

definition *U* where U = (λa b::real. uniform-measure lborel {a..b})

declare *U-def*[simp]

fun *insR*:: 'a::linorder ⇒ ('a × real) tree ⇒ 'a set ⇒ ('a × real) tree measure
where
insR x t A = distr (U 0 1) (tree-sigma (count-space A ⊗_M lborel)) (λp. ins x p t)

fun *rinss* :: 'a::linorder list ⇒ ('a × real) tree ⇒ 'a set ⇒ ('a × real) tree measure
where
rinss [] t A = return (tree-sigma (count-space A ⊗_M lborel)) t |
rinss (x#xs) t A = *insR* x t A ≫ (λt. *rinss* xs t A)

lemma *sets-rinss'*:

assumes countable B set ys ⊆ B
shows t ∈ trees (B × UNIV) ⇒ sets (rinss ys t B) = sets (tree-sigma (count-space B ⊗_M lborel))
 ⟨proof⟩

lemma *measurable-foldl* [measurable]:

assumes f ∈ A →_M B set xs ⊆ space C
assumes ∧c. c ∈ set xs ⇒ (λ(a,b). g a b c) ∈ (A ⊗_M B) →_M B
shows (λx. foldl (g x) (f x) xs) ∈ A →_M B
 ⟨proof⟩

lemma *ins-trees*: $t \in \text{trees } A \implies (x,y) \in A \implies \text{ins } x \ y \ t \in \text{trees } A$
 ⟨proof⟩

5.2 Main result

In our setting, we have some countable set of values that may appear in the input and a concrete list consisting only of those elements with no repeated elements.

We further define an abbreviation for the uniform distribution of permutations of that lists.

context

fixes $xs::'a::\text{linorder list}$ **and** $A::'a \text{ set}$ **and** $\text{random-perm} :: 'a \text{ list} \Rightarrow 'a \text{ list measure}$

assumes *con-assms*: *countable* A *set* $xs \subseteq A$ *distinct* xs

defines $\text{random-perm} \equiv (\lambda xs. \text{uniform-measure } (\text{count-space } (\text{permutations-of-set } (\text{set } xs))))$
($\text{permutations-of-set } (\text{set } xs)$)

begin

Again, we first need some facts about measurability.

lemma *sets-rinss* [*simp*]:

assumes $t \in \text{trees } (A \times \text{UNIV})$

shows $\text{sets } (\text{rinss } xs \ t \ A) = \text{tree-sigma } (\text{count-space } A \otimes_M \text{borel})$

⟨proof⟩

lemma *bst-of-list-measurable* [*measurable*]:

$\text{bst-of-list} \in \text{measurable } (\text{count-space } (\text{lists } A)) (\text{tree-sigma } (\text{count-space } A))$

⟨proof⟩

lemma *insort-wrt-measurable* [*measurable*]:

$(\lambda x. \text{insort-wrt } x \ xs) \in \text{count-space } (\text{Pow } (A \times A)) \rightarrow_M \text{count-space } (\text{lists } A)$

⟨proof⟩

lemma *bst-of-list-sort-measurable* [*measurable*]:

$(\lambda x. \text{bst-of-list } (\text{sort-key } x \ xs)) \in$

$\text{Pi}_M (\text{set } xs) (\lambda i. \text{borel}::\text{real measure}) \rightarrow_M \text{tree-sigma } (\text{count-space } A)$

⟨proof⟩

In a first step, we convert the bulk insertion operation to first choosing the priorities i. i. d. ahead of time and then inserting all the elements deterministically with their associated priority.

lemma *random-treap-fold*:

assumes $t \in \text{space } (\text{tree-sigma } (\text{count-space } A \otimes_M \text{lborel}))$

shows $\text{rinss } xs \ t \ A = \text{distr } (\text{Pi}_M x \in \text{set } xs. \mathcal{U} \ 0 \ 1)$
($\text{tree-sigma } (\text{count-space } A \otimes_M \text{lborel})$)
($\lambda p. \text{foldl } (\lambda t \ x. \text{ins } x \ (p \ x) \ t) \ t \ xs$)

⟨proof⟩

corollary *random-treap-fold-Leaf*:

shows $\text{rinss } xs \text{ Leaf } A =$
 $\text{distr } (\prod_M x \in \text{set } xs. \mathcal{U} \ 0 \ 1)$
 $(\text{tree-sigma } (\text{count-space } A \otimes_M \text{lborel}))$
 $(\lambda p. \text{foldl } (\lambda t x. \text{ins } x (p \ x) \ t) \ \text{Leaf } xs)$
 $\langle \text{proof} \rangle$

Next, we show that additionally forgetting the priorities in the end will yield the same distribution as inserting the elements into a BST by ascending priority.

lemma *rinss-bst-of-list*:

$\text{distr } (\text{rinss } xs \ \text{Leaf } A) (\text{tree-sigma } (\text{count-space } A)) (\text{map-tree } fst) =$
 $\text{distr } (Pi_M (\text{set } xs) (\lambda x. \mathcal{U} \ 0 \ 1)) (\text{tree-sigma } (\text{count-space } A))$
 $(\lambda p. \text{bst-of-list } (\text{sort-key } p \ xs)) (\text{is } ?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

This in turn is the same as choosing a random permutation of the input list and inserting the elements into a BST in that order.

lemma *lborel-permutations-of-set-bst-of-list*:

shows $\text{distr } (Pi_M (\text{set } xs) (\lambda x. \mathcal{U} \ 0 \ 1)) (\text{tree-sigma } (\text{count-space } A))$
 $(\lambda p. \text{bst-of-list } (\text{sort-key } p \ xs)) =$
 $\text{distr } (\text{random-perm } xs) (\text{tree-sigma } (\text{count-space } A)) \text{bst-of-list } (\text{is } ?lhs =$
 $?rhs)$
 $\langle \text{proof} \rangle$

lemma *distr-bst-of-list-tree-sigma-count-space*:

$\text{distr } (\text{random-perm } xs) (\text{tree-sigma } (\text{count-space } A)) \text{bst-of-list} =$
 $\text{distr } (\text{random-perm } xs) (\text{count-space } (\text{trees } A)) \text{bst-of-list}$
 $\langle \text{proof} \rangle$

This is the same as a *random BST*.

lemma *distr-bst-of-list-random-bst*:

$\text{distr } (\text{random-perm } xs) (\text{count-space } (\text{trees } A)) \text{bst-of-list} =$
 $\text{restrict-space } (\text{random-bst } (\text{set } xs)) (\text{trees } A) (\text{is } ?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

We put everything together and obtain our main result:

theorem *rinss-random-bst*:

$\text{distr } (\text{rinss } xs \ \langle \rangle \ A) (\text{tree-sigma } (\text{count-space } A)) (\text{map-tree } fst) =$
 $\text{restrict-space } (\text{measure-pmf } (\text{random-bst } (\text{set } xs))) (\text{trees } A)$
 $\langle \text{proof} \rangle$

end

end

References

- [1] M. Eberl, M. Haslbeck, and T. Nipkow. Verified analysis of random trees, 2018 (forthcoming).
- [2] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4):464–497, Oct 1996.