The Fisher–Yates shuffle

Manuel Eberl

March 17, 2025

Abstract

This work defines and proves the correctness of the Fisher–Yates shuffle [1, 2, 3] for shuffling – i. e. producing a random permutation – of a list. The algorithm proceeds by traversing the list and in each step swapping the current element with a random element from the remaining list.

Contents

1 Fisher–Yates shuffle

theory *Fisher-Yates*
  imports *HOL−Probability.Probability*
begin

lemma *integral-pmf-of-multiset*:
  $A \neq \{\#\} \Longrightarrow (\int x.\ (f\ x :: real)\ \partial measure\text{-}pmf\ (pmf\text{-}of\text{-}multiset\ A)) =$
    $(\sum x \in set\text{-}mset\ A.\ of\text{-}nat\ (count\ A\ x) * f\ x)\ /\ of\text{-}nat\ (size\ A)$
  ⟨*proof*⟩

lemma *pmf-bind-pmf-of-multiset*:
  $A \neq \{\#\} \Longrightarrow pmf\ (pmf\text{-}of\text{-}multiset\ A \gg= f)\ y =$
    $(\sum x \in set\text{-}mset\ A.\ real\ (count\ A\ x) * pmf\ (f\ x)\ y)\ /\ real\ (size\ A)$
  ⟨*proof*⟩

lemma *pmf-map-inj-inv*:
  assumes $inj\text{-}on\ f\ (set\text{-}pmf\ p)$
  assumes $\bigwedge x.\ f'\ (f\ x) = x$
  shows $pmf\ (map\text{-}pmf\ f\ p)\ x = (if\ x \in range\ f\ then\ pmf\ p\ (f'\ x)\ else\ 0)$
⟨*proof*⟩

1.1 Swapping elements in a list

definition *swap* **where** $swap\ xs\ i\ j = xs[i := xs!j,\ j := xs\ !\ i]$

lemma *length-swap* [*simp*]: $length\ (swap\ xs\ i\ j) = length\ xs$
  ⟨*proof*⟩

lemma *swap-eq-Nil-iff* [*simp*]: $swap\ xs\ i\ j = [] \longleftrightarrow xs = []$
  ⟨*proof*⟩

lemma *nth-swap*: $i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow$
    $swap\ xs\ i\ j\ !\ k = (if\ k = i\ then\ xs\ !\ j\ else\ if\ k = j\ then\ xs\ !\ i\ else\ xs\ !\ k)$
  ⟨*proof*⟩

lemma *map-swap*: $i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow map\ f\ (swap\ xs\ i\ j) = swap\ (map\ f\ xs)\ i\ j$
  ⟨*proof*⟩

lemma *swap-swap*: $i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow swap\ (swap\ xs\ i\ j)\ j\ i = xs$
  ⟨*proof*⟩

lemma *mset-swap*: $i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow mset\ (swap\ xs\ i\ j) = mset\ xs$
  ⟨*proof*⟩

lemma *hd-swap-0*: $i < \mathit{length}\ xs \Longrightarrow \mathit{hd}\ (\mathit{swap}\ xs\ 0\ i) = xs\ !\ i$
⟨*proof*⟩

1.2 Random Permutations

First, we prove the intuitively obvious fact that choosing a random permutation of a multiset can be done by first randomly choosing the first element and then randomly choosing the rest of the list.

lemma *pmf-of-set-permutations-of-multiset-nonempty*:
assumes $(A :: 'a\ \mathit{multiset}) \neq \{\#\}$
shows $\mathit{pmf\text{-}of\text{-}set}\ (\mathit{permutations\text{-}of\text{-}multiset}\ A) =$
$\mathit{do}\ \{x \leftarrow \mathit{pmf\text{-}of\text{-}multiset}\ A;$
$xs \leftarrow \mathit{pmf\text{-}of\text{-}set}\ (\mathit{permutations\text{-}of\text{-}multiset}\ (A - \{\#x\#\}));$
$\mathit{return\text{-}pmf}\ (x\#xs)$
$\}$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

1.3 Shuffling Lists

We define shuffling of a list as choosing from the set of all lists that correspond to the same multiset uniformly at random.

definition $\mathit{shuffle} :: 'a\ \mathit{list} \Rightarrow 'a\ \mathit{list}\ \mathit{pmf}$ **where**
$\mathit{shuffle}\ xs = \mathit{pmf\text{-}of\text{-}set}\ (\mathit{permutations\text{-}of\text{-}multiset}\ (\mathit{mset}\ xs))$

lemma *shuffle-empty* [*simp*]: $\mathit{shuffle}\ [] = \mathit{return\text{-}pmf}\ []$
⟨*proof*⟩

lemma *shuffle-singleton* [*simp*]: $\mathit{shuffle}\ [x] = \mathit{return\text{-}pmf}\ [x]$
⟨*proof*⟩

The crucial ingredient of the Fisher–Yates shuffle is the following lemma, which decomposes a shuffle into swapping the first element of the list with a random element of the remaining list and shuffling the new remaining list.

With a random-access implementation of a list – such as an array – all of the required operations are cheap and the resulting algorithm runs in linear time.

lemma *shuffle-fisher-yates-step*:
assumes *xs-nonempty* [*simp*]: $xs \neq []$
shows $\mathit{shuffle}\ xs =$
$\mathit{do}\ \{i \leftarrow \mathit{pmf\text{-}of\text{-}set}\ \{..<\mathit{length}\ xs\};$
$\mathit{let}\ ys = \mathit{swap}\ xs\ 0\ i;$
$zs \leftarrow \mathit{shuffle}\ (\mathit{tl}\ ys);$
$\mathit{return\text{-}pmf}\ (\mathit{hd}\ ys\ \#\ zs)$
$\}$
⟨*proof*⟩

1.4 Forward Fisher-Yates Shuffle

The actual Fisher–Yates shuffle is now merely a kind of tail-recursive version of decomposition described above. Note that unlike the traditional Fisher–Yates shuffle, we shuffle the list from front to back, which is the more natural way to do it when working with linked lists.

function *fisher-yates-aux* **where**
  fisher-yates-aux i xs = (*if i* + *1* ≥ *length xs then return-pmf xs else*
    do {*j* ← *pmf-of-set* {*i..<length xs*};
      fisher-yates-aux (*i* + *1*) (*swap xs i j*)})
⟨*proof*⟩
termination ⟨*proof*⟩

declare *fisher-yates-aux.simps* [*simp del*]

lemma *fisher-yates-aux-correct*:
  fisher-yates-aux i xs = *map-pmf* (*λys. take i xs* @ *ys*) (*shuffle* (*drop i xs*))
⟨*proof*⟩

definition *fisher-yates* **where**
  fisher-yates = *fisher-yates-aux 0*

lemma *fisher-yates-correct*: *fisher-yates xs* = *shuffle xs*
  ⟨*proof*⟩

1.5 Backwards Fisher-Yates Shuffle

We can now easily derive the classical Fisher–Yates shuffle, which goes through the list from back to front and show its equivalence to the forward Fisher–Yates shuffle.

fun *fisher-yates-alt-aux* **where**
  fisher-yates-alt-aux i xs = (*if i* = *0 then return-pmf xs else*
    do {*j* ← *pmf-of-set* {*..i*};
      fisher-yates-alt-aux (*i* − *1*) (*swap xs i j*)})

declare *fisher-yates-alt-aux.simps* [*simp del*]

lemma *fisher-yates-alt-aux-altdef*:
  i < *length xs* ⟹ *fisher-yates-alt-aux i xs* =
    map-pmf rev (*fisher-yates-aux* (*length xs* − *i* − *1*) (*rev xs*))
⟨*proof*⟩

definition *fisher-yates-alt* **where**
  fisher-yates-alt xs = *fisher-yates-alt-aux* (*length xs* − *1*) *xs*

lemma *fisher-yates-alt-aux-correct*:
  fisher-yates-alt xs = *shuffle xs*
⟨*proof*⟩

1.6 Code generation test

Isabelle's code generator allows us to produce executable code both for *shuffle* and for *fisher-yates* and *fisher-yates-alt*. However, this code does not produce a random sample (i.e. a single randomly permuted list) – which is, in fact, the only purpose of the Fisher–Yates algorithm – but the entire probability distribution consisting of $n!$ lists, each with probability $1/n!$.

In the future, it would be nice if Isabelle also had some code generation facility that supports generating sampling code.

value [*code*] *shuffle ''abcd''*
value [*code*] *fisher-yates ''abcd''*
value [*code*] *fisher-yates-alt ''abcd''*

end

References

[1] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research*, pages 26–27. Oliver & Boyd, Third edition, 1948.

[2] D. E. Knuth. In *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Third edition, 1997.

[3] Wikipedia. Fisher–Yates shuffle – Wikipedia, the free encyclopedia, 2016. [Online; accessed 5 October 2016].