Linear orders as rankings

Manuel Eberl

January 23, 2026

This entry formalises the obvious isomorphism between finite linear orders and lists, where the list in question is interpreted as a *ranking*, i.e. it lists the elements in descending order without repetition.

It also provides an executable algorithm to compute topological sortings, i.e. all rankings whose linear orders are extensions of a given relation.

Contents

1 Rankings

theory *Rankings*
imports
  HOL−Combinatorics.Multiset-Permutations
  List−Index.List-Index
  Randomised-Social-Choice.Order-Predicates
begin

1.1 Preliminaries

lemma *find-index-map*: *find-index* P (*map* f *xs*) = *find-index* ($\lambda x.$ P (f x)) *xs*
  ⟨*proof*⟩

lemma *map-index-self*:
  assumes *distinct xs*
  shows *map* (*index xs*) *xs* = [*0..<length xs*]
⟨*proof*⟩

lemma *bij-betw-map-prod*:
  assumes *bij-betw* f A B *bij-betw* g C D
  shows *bij-betw* (*map-prod* f g) ($A \times C$) ($B \times D$)
  ⟨*proof*⟩

definition *comap-relation* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *relation* $\Rightarrow 'b$ *relation* **where**
  comap-relation f R = (λx y. $\exists x'$ y'. $x = f$ $x' \wedge y = f$ $y' \wedge R$ x' y')

lemma *is-weak-ranking-map-singleton-iff* [*simp*]:
  is-weak-ranking (*map* ($\lambda x.$ $\{x\}$) *xs*) $\longleftrightarrow$ *distinct xs*
  ⟨*proof*⟩

lemma *is-finite-weak-ranking-map-singleton-iff* [*simp*]:
  is-finite-weak-ranking (*map* ($\lambda x.$ $\{x\}$) *xs*) $\longleftrightarrow$ *distinct xs*
  ⟨*proof*⟩

lemma *of-weak-ranking-altdef'*:
  assumes *is-weak-ranking xs*

shows $\textit{of-weak-ranking xs x y} \longleftrightarrow x \in \bigcup(\textit{set xs}) \wedge y \in \bigcup(\textit{set xs}) \wedge$
$\textit{find-index } ((\in)\ x)\ \textit{xs} \geq \textit{find-index } ((\in)\ y)\ \textit{xs}$
⟨*proof*⟩

1.2 Definition

A *ranking* is a representation of a linear order on a finite set as a list in descending order, starting with the biggest element. Clearly, this gives a bijection between the linear orders on a finite set and the permutations of that set.

inductive *of-ranking* :: *'alt list* ⇒ *'alt relation* **where**
$i \leq j \Longrightarrow i < \textit{length xs} \Longrightarrow j < \textit{length xs} \Longrightarrow \ \textit{xs} \mathbin{!} i \succeq[\textit{of-ranking xs}]\ \textit{xs} \mathbin{!} j$

lemma *of-ranking-conv-of-weak-ranking*:
$x \succeq[\textit{of-ranking xs}]\ y \longleftrightarrow x \succeq[\textit{of-weak-ranking } (\textit{map } (\lambda x.\ \{x\})\ \textit{xs})]\ y$
⟨*proof*⟩

lemma *of-ranking-imp-in-set*:
assumes *of-ranking xs a b*
shows $a \in \textit{set xs}\ b \in \textit{set xs}$
⟨*proof*⟩

lemma *of-ranking-Nil* [*simp*]: $\textit{of-ranking } [] = (\lambda\text{- -}.\ \textit{False})$
⟨*proof*⟩

lemma *of-ranking-Nil'* [*code*]: $\textit{of-ranking } []\ x\ y = \textit{False}$
⟨*proof*⟩

lemma *of-ranking-Cons* [*code*]:
$x \succeq[\textit{of-ranking } (z\#zs)]\ y \longleftrightarrow x = z \wedge y \in \textit{set } (z\#zs) \vee x \succeq[\textit{of-ranking zs}]\ y$
⟨*proof*⟩

lemma *of-ranking-Cons'*:
assumes $\textit{distinct } (x\#xs)\ a \in \textit{set } (x\#xs)\ b \in \textit{set } (x\#xs)$
shows $\textit{of-ranking } (x\#xs)\ a\ b \longleftrightarrow b = x \vee (a \neq x \wedge \textit{of-ranking xs a b})$
⟨*proof*⟩

lemma *of-ranking-append*:
$x \succeq[\textit{of-ranking } (xs\ @\ ys)]\ y \longleftrightarrow x \in \textit{set xs} \wedge y \in \textit{set ys} \vee x \succeq[\textit{of-ranking xs}]\ y \vee x \succeq[\textit{of-ranking ys}]\ y$
⟨*proof*⟩

lemma *of-ranking-strongly-preferred-Cons-iff*:
assumes $\textit{distinct } (x\ \#\ xs)$
shows $a \succ[\textit{of-ranking } (x\ \#\ xs)]\ b \longleftrightarrow x = a \wedge b \in \textit{set xs} \vee a \succ[\textit{of-ranking xs}]\ b$
⟨*proof*⟩

lemma *of-ranking-strongly-preferred-append-iff*:
assumes $\textit{distinct } (xs\ @\ ys)$
shows $a \succ[\textit{of-ranking } (xs\ @\ ys)]\ b \longleftrightarrow$

$a \in \mathit{set}\ \mathit{xs} \wedge b \in \mathit{set}\ \mathit{ys} \vee a \succ[\mathit{of\text{-}ranking}\ \mathit{xs}]\ b \vee a \succ[\mathit{of\text{-}ranking}\ \mathit{ys}]\ b$
⟨*proof*⟩

lemma *not-strongly-preferred-of-ranking-iff*:
assumes $a \in \mathit{set}\ \mathit{xs}\ b \in \mathit{set}\ \mathit{xs}$
shows $\neg a \prec[\mathit{of\text{-}ranking}\ \mathit{xs}]\ b \longleftrightarrow a \succeq[\mathit{of\text{-}ranking}\ \mathit{xs}]\ b$
⟨*proof*⟩

lemma *of-ranking-refl*:
assumes $x \in \mathit{set}\ \mathit{xs}$
shows $x \preceq[\mathit{of\text{-}ranking}\ \mathit{xs}]\ x$
⟨*proof*⟩

lemma *of-ranking-altdef*:
assumes $\mathit{distinct}\ \mathit{xs}\ x \in \mathit{set}\ \mathit{xs}\ y \in \mathit{set}\ \mathit{xs}$
shows $\mathit{of\text{-}ranking}\ \mathit{xs}\ x\ y \longleftrightarrow \mathit{index}\ \mathit{xs}\ x \geq \mathit{index}\ \mathit{xs}\ y$
⟨*proof*⟩

lemma *of-ranking-altdef'*:
assumes $\mathit{distinct}\ \mathit{xs}$
shows $\mathit{of\text{-}ranking}\ \mathit{xs}\ x\ y \longleftrightarrow x \in \mathit{set}\ \mathit{xs} \wedge y \in \mathit{set}\ \mathit{xs} \wedge \mathit{index}\ \mathit{xs}\ x \geq \mathit{index}\ \mathit{xs}\ y$
⟨*proof*⟩

lemma *of-ranking-nth-iff*:
assumes $\mathit{distinct}\ \mathit{xs}\ i < \mathit{length}\ \mathit{xs}\ j < \mathit{length}\ \mathit{xs}$
shows $\mathit{of\text{-}ranking}\ \mathit{xs}\ (\mathit{xs}\ !\ i)\ (\mathit{xs}\ !\ j) \longleftrightarrow i \geq j$
⟨*proof*⟩

lemma *strongly-preferred-of-ranking-nth-iff*:
assumes $\mathit{distinct}\ \mathit{xs}\ i < \mathit{length}\ \mathit{xs}\ j < \mathit{length}\ \mathit{xs}$
shows $\mathit{xs}\ !\ i \succ[\mathit{of\text{-}ranking}\ \mathit{xs}]\ \mathit{xs}\ !\ j \longleftrightarrow i < j$
⟨*proof*⟩

lemma *of-ranking-total*: $x \in \mathit{set}\ \mathit{xs} \Longrightarrow y \in \mathit{set}\ \mathit{xs} \Longrightarrow \mathit{of\text{-}ranking}\ \mathit{xs}\ x\ y \vee \mathit{of\text{-}ranking}\ \mathit{xs}\ y\ x$
⟨*proof*⟩

lemma *of-ranking-antisym*:
$x \in \mathit{set}\ \mathit{xs} \Longrightarrow y \in \mathit{set}\ \mathit{xs} \Longrightarrow \mathit{of\text{-}ranking}\ \mathit{xs}\ x\ y \Longrightarrow \mathit{of\text{-}ranking}\ \mathit{xs}\ y\ x \Longrightarrow \mathit{distinct}\ \mathit{xs} \Longrightarrow x = y$
⟨*proof*⟩

lemma *finite-linorder-of-ranking*:
assumes $\mathit{set}\ \mathit{xs} = A\ \mathit{distinct}\ \mathit{xs}$
shows $\mathit{finite\text{-}linorder\text{-}on}\ A\ (\mathit{of\text{-}ranking}\ \mathit{xs})$
⟨*proof*⟩

lemma *linorder-of-ranking*:
assumes $\mathit{set}\ \mathit{xs} = A\ \mathit{distinct}\ \mathit{xs}$

shows *linorder-on A* (*of-ranking xs*)
⟨*proof*⟩

lemma *total-preorder-of-ranking*:
assumes *set xs* = *A distinct xs*
shows *total-preorder-on A* (*of-ranking xs*)
⟨*proof*⟩

1.3 Transformations

lemma *map-relation-of-ranking*:
map-relation f (*of-ranking xs*) = *of-weak-ranking* (*map* ($\lambda x.$ *f* −‘ {*x*}) *xs*)
⟨*proof*⟩

lemma *of-ranking-map*: *of-ranking* (*map f xs*) = *comap-relation f* (*of-ranking xs*)
⟨*proof*⟩

lemma *of-ranking-permute′*:
assumes *f permutes set xs*
shows *map-relation f* (*of-ranking xs*) = *of-ranking* (*map* (*inv f*) *xs*)
⟨*proof*⟩

lemma *of-ranking-permute*:
assumes *f permutes set xs*
shows *of-ranking* (*map f xs*) = *map-relation* (*inv f*) (*of-ranking xs*)
⟨*proof*⟩

lemma *of-ranking-rev* [*simp*]:
of-ranking (*rev xs*) *x y* ⟷ *of-ranking xs y x*
⟨*proof*⟩

lemma *of-ranking-filter*:
of-ranking (*filter P xs*) = *restrict-relation* {*x*. *P x*} (*of-ranking xs*)
⟨*proof*⟩

lemma *strongly-preferred-of-ranking-conv-index*:
assumes *distinct xs*
shows *x* ≺[*of-ranking xs*] *y* ⟷ *x* ∈ *set xs* ∧ *y* ∈ *set xs* ∧ *index xs x* > *index xs y*
⟨*proof*⟩

lemma *restrict-relation-of-weak-ranking-Cons*:
assumes *distinct* (*x* # *xs*)
shows *restrict-relation* (*set xs*) (*of-ranking* (*x* # *xs*)) = *of-ranking xs*
⟨*proof*⟩

lemma *of-ranking-zero-upt-nat*:
of-ranking [*0*::*nat*..<*n*] = ($\lambda x\ y.\ x \geq y \wedge x < n$)
⟨*proof*⟩

lemma *of-ranking-rev-zero-upt-nat*:
  of-ranking (*rev* [*0*::*nat*..<*n*]) = ($\lambda x\ y.\ x \leq y \wedge y < n$)
  ⟨*proof*⟩

lemma *sorted-wrt-ranking*: *distinct xs* $\Longrightarrow$ *sorted-wrt* (*of-ranking xs*) (*rev xs*)
  ⟨*proof*⟩

1.4 Inverse operation and isomorphism

lemma (**in** *finite-linorder-on*) *of-ranking-ranking*: *of-ranking* (*ranking le*) = *le*
⟨*proof*⟩

lemma (**in** *finite-linorder-on*) *distinct-ranking*: *distinct* (*ranking le*)
  ⟨*proof*⟩

lemma *ranking-of-ranking*:
  assumes *distinct xs*
  shows   *ranking* (*of-ranking xs*) = *xs*
⟨*proof*⟩

lemma (**in** *finite-linorder-on*) *set-ranking*: *set* (*ranking le*) = *carrier*
  ⟨*proof*⟩

lemma *bij-betw-permutations-of-set-finite-linorders-on*:
  bij-betw of-ranking (*permutations-of-set A*) {*R*. *finite-linorder-on A R*}
  ⟨*proof*⟩

lemma *bij-betw-permutations-of-set-finite-linorders-on′*:
  bij-betw ranking {*R*. *finite-linorder-on A R*} (*permutations-of-set A*)
  ⟨*proof*⟩

lemma *card-linorders-on*:
  assumes *finite A*
  shows   *card* {*R*. *linorder-on A R*} = *fact* (*card A*)
⟨*proof*⟩

lemma *finite-linorders-on* [*intro*]:
  assumes *finite A*
  shows   *finite* {*R*. *linorder-on A R*}
⟨*proof*⟩

end

1.5 Topological sorting

theory *Topological-Sortings-Rankings*
  imports *Rankings*
begin

The following returns the set of all rankings of the given set A that are extensions of the given relation R, i.e. all topological sortings of R.

Note that there are no requirements about R; in particular it does not have to be reflexive, antisymmetric, or transitive. If it is not antisymmetric or not transitive, the result set will simply be empty.

function *topo-sorts* :: $'a\ set \Rightarrow\ 'a\ relation \Rightarrow\ 'a\ list\ set$ **where**
  $topo\text{-}sorts\ A\ R =$
    $(if\ infinite\ A\ then\ \{\}\ else\ if\ A = \{\}\ then\ \{[]\}\ else$
    $\bigcup x{\in}\{x{\in}A.\ \forall z{\in}A.\ R\ x\ z \longrightarrow z = x\}.\ (\lambda xs.\ x\ \#\ xs)\ `\ topo\text{-}sorts\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x))$
  ⟨*proof*⟩
termination
⟨*proof*⟩

lemmas [*simp del*] = *topo-sorts.simps*

lemma *topo-sorts-empty* [*simp*]: $topo\text{-}sorts\ \{\}\ R = \{[]\}$
  ⟨*proof*⟩

lemma *topo-sorts-infinite*: $infinite\ A \Longrightarrow topo\text{-}sorts\ A\ R = \{\}$
  ⟨*proof*⟩

lemma *topo-sorts-rec*:
  $finite\ A \Longrightarrow A \neq \{\} \Longrightarrow$
    $topo\text{-}sorts\ A\ R = (\bigcup x{\in}\{x{\in}A.\ \forall z{\in}A.\ R\ x\ z \longrightarrow z = x\}.$
      $(\lambda xs.\ x\ \#\ xs)\ `\ topo\text{-}sorts\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x))$
  ⟨*proof*⟩

lemma *topo-sorts-cong* [*cong*]:
  assumes $A = B$ $\bigwedge x\ y.\ x \in A \Longrightarrow y \in B \Longrightarrow x \neq y \Longrightarrow R\ x\ y = R'\ x\ y$
  shows $topo\text{-}sorts\ A\ R = topo\text{-}sorts\ B\ R'$
⟨*proof*⟩

lemma *topo-sorts-correct*:
  assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow x \in A \wedge y \in A$
  shows $topo\text{-}sorts\ A\ R = \{xs{\in}permutations\text{-}of\text{-}set\ A.\ R \leq of\text{-}ranking\ xs\}$
  ⟨*proof*⟩

lemma *topo-sorts-nonempty*:
  assumes $finite\ A$ $\bigwedge x\ y.\ R\ x\ y \Longrightarrow x \in A \wedge y \in A$ $\bigwedge x\ y.\ R\ x\ y \Longrightarrow \neg R\ y\ x$ $transp\ R$
  shows $topo\text{-}sorts\ A\ R \neq \{\}$
  ⟨*proof*⟩

lemma *bij-betw-topo-sorts-linorders-on*:
  assumes $\bigwedge x\ y.\ R\ x\ y \Longrightarrow x \in A \wedge y \in A$
  shows $bij\text{-}betw\ of\text{-}ranking\ (topo\text{-}sorts\ A\ R)\ \{R'.\ finite\text{-}linorder\text{-}on\ A\ R' \wedge R \leq R'\}$
⟨*proof*⟩

In the following, we give a more convenient formulation of this for computation.

The input is a relation represented as a list of pairs (x, ys) where *ys* is the set of all elements such that (x, y) is in the relation.

function *topo-sorts-aux* :: (*'a* × *'a set*) *list* ⇒ *'a list list* **where**
  topo-sorts-aux xs =
    (*if xs* = [] *then* [[]] *else*
     List.bind (*map fst* (*filter* (λ(-,*ys*). *ys* = {}) *xs*))
       (λ*x*. *map* ((#) *x*) (*topo-sorts-aux*
         (*map* (*map-prod id* (*Set.filter* (λ*y*. *y* ≠ *x*))) (*filter* (λ(*y*,-). *y* ≠ *x*) *xs*)))))
  ⟨*proof*⟩
termination
  ⟨*proof*⟩

lemmas [*simp del*] = *topo-sorts-aux.simps*

lemma *topo-sorts-aux-Nil* [*simp*]: *topo-sorts-aux* [] = [[]]
  ⟨*proof*⟩

lemma *topo-sorts-aux-rec*:
  xs ≠ [] ⟹ *topo-sorts-aux xs* =
    List.bind (*map fst* (*filter* (λ(-,*ys*). *ys* = {}) *xs*))
      (λ*x*. *map* ((#) *x*) (*topo-sorts-aux*
        (*map* (*map-prod id* (*Set.filter* (λ*y*. *y* ≠ *x*))) (*filter* (λ(*y*,-). *y* ≠ *x*) *xs*))))
  ⟨*proof*⟩

lemma *topo-sorts-aux-Cons*:
  topo-sorts-aux (*y*#*xs*) =
    List.bind (*map fst* (*filter* (λ(-,*ys*). *ys* = {}) (*y*#*xs*)))
      (λ*x*. *map* ((#) *x*) (*topo-sorts-aux*
        (*map* (*map-prod id* (*Set.filter* (λ*y*. *y* ≠ *x*))) (*filter* (λ(*y*,-). *y* ≠ *x*) (*y*#*xs*)))))
  ⟨*proof*⟩

lemma *set-topo-sorts-aux*:
  assumes *distinct* (*map fst xs*)
  assumes ⋀*x ys*. (*x*, *ys*) ∈ *set xs* ⟹ *ys* ⊆ *set* (*map fst xs*) − {*x*}
  shows *set* (*topo-sorts-aux xs*) =
      topo-sorts (*set* (*map fst xs*)) (λ*x y*. ∃ *ys*. (*x*, *ys*) ∈ *set xs* ∧ *y* ∈ *ys*)
  ⟨*proof*⟩

lemma *topo-sorts-code* [*code*]:
  topo-sorts (*set xs*) *R* = (*let xs'* = *remdups xs in*
    set (*topo-sorts-aux* (*map* (λ*x*. (*x*, *set* (*filter* (λ*y*. *y* ≠ *x* ∧ *R x y*) *xs'*))) *xs'*)))
⟨*proof*⟩

end