Linear orders as rankings

Manuel Eberl

February 6, 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* (λ*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* × *C*) (*B* × *D*)
⟨*proof*⟩

definition *comap-relation* :: (′*a* ⇒ ′*b*) ⇒ ′*a relation* ⇒ ′*b relation* **where**
comap-relation f R = (λ*x y*. ∃ *x′ y′*. *x* = *f x′* ∧ *y* = *f y′* ∧ *R x′ y′*)

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

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

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

shows *of-weak-ranking xs x y* $\longleftrightarrow$ $x \in \bigcup(set\ xs) \wedge y \in \bigcup(set\ xs) \wedge$
find-index $((\in)\ x)\ xs \geq$ *find-index* $((\in)\ y)\ xs$
⟨*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**
$i \leq j \Longrightarrow i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow\ xs\ !\ i \succeq[of\text{-}ranking\ xs]\ xs\ !\ j$

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

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

lemma *of-ranking-Nil* [*simp*]: *of-ranking* [] = (λ- -. *False*)
⟨*proof*⟩

lemma *of-ranking-Nil'* [*code*]: *of-ranking* [] *x y* = *False*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *linorder-of-ranking*:
assumes *set xs* $= A$ *distinct 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 (λx. 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] = (λx y. x ≥ y ∧ x < n)
⟨*proof*⟩

lemma *of-ranking-rev-zero-upt-nat*:
of-ranking (*rev* [*0*::*nat*..<*n*]) = (λ*x y*. *x* ≤ *y* ∧ *y* < *n*)
⟨*proof*⟩

lemma *sorted-wrt-ranking*: *distinct xs* ⟹ *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* ⇒ *'a relation* ⇒ *'a list set* **where**
topo-sorts A R =
(*if infinite A then* {} *else if A* = {} *then* {[]} *else*
⋃*x*∈{*x*∈*A*. ∀*z*∈*A*. *R x z* ⟶ *z* = *x*}. (λ*xs*. *x* # *xs*) ‘ *topo-sorts* (*A* − {*x*}) (λ*y z*. *R y z* ∧ *y* ≠ *x* ∧ *z* ≠ *x*))
⟨*proof*⟩
termination
⟨*proof*⟩

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

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

lemma *topo-sorts-infinite*: *infinite A* ⟹ *topo-sorts A R* = {}
⟨*proof*⟩

lemma *topo-sorts-rec*:
finite A ⟹ *A* ≠ {} ⟹
topo-sorts A R = (⋃*x*∈{*x*∈*A*. ∀*z*∈*A*. *R x z* ⟶ *z* = *x*}.
(λ*xs*. *x* # *xs*) ‘ *topo-sorts* (*A* − {*x*}) (λ*y z*. *R y z* ∧ *y* ≠ *x* ∧ *z* ≠ *x*))
⟨*proof*⟩

lemma *topo-sorts-cong* [*cong*]:
assumes *A* = *B* ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *B* ⟹ *x* ≠ *y* ⟹ *R x y* = *R*′ *x y*
shows *topo-sorts A R* = *topo-sorts B R*′
⟨*proof*⟩

lemma *topo-sorts-correct*:
assumes ⋀*x y*. *R x y* ⟹ *x* ∈ *A* ∧ *y* ∈ *A*
shows *topo-sorts A R* = {*xs*∈*permutations-of-set A*. *R* ≤ *of-ranking xs*}
⟨*proof*⟩

lemma *topo-sorts-nonempty*:
assumes *finite A* ⋀*x y*. *R x y* ⟹ *x* ∈ *A* ∧ *y* ∈ *A* ⋀*x y*. *R x y* ⟹ ¬*R y x transp R*
shows *topo-sorts A R* ≠ {}
⟨*proof*⟩

lemma *bij-betw-topo-sorts-linorders-on*:
assumes ⋀*x y*. *R x y* ⟹ *x* ∈ *A* ∧ *y* ∈ *A*
shows *bij-betw of-ranking* (*topo-sorts A R*) {*R*′. *finite-linorder-on A R*′ ∧ *R* ≤ *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* :: $('a \times 'a\ set)\ list \Rightarrow 'a\ list\ list$ **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