

# Linear orders as rankings

Manuel Eberl

April 12, 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

<b>1</b>	<b>Rankings</b>	<b>2</b>
1.1	Preliminaries . . . . .	2
1.2	Definition . . . . .	2
1.3	Transformations . . . . .	5
1.4	Inverse operation and isomorphism . . . . .	6
1.5	Topological sorting . . . . .	6

## 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*:  $\text{find-index } P \ (\text{map } f \ xs) = \text{find-index } (\lambda x. P \ (f \ x)) \ xs$   
*<proof>*

**lemma** *map-index-self*:  
**assumes** *distinct xs*  
**shows**  $\text{map } (\text{index } xs) \ xs = [0..<\text{length } xs]$   
*<proof>*

**lemma** *bij-betw-map-prod*:  
**assumes** *bij-betw f A B bij-betw g C D*  
**shows**  $\text{bij-betw } (\text{map-prod } f \ g) \ (A \times C) \ (B \times D)$   
*<proof>*

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

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

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

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

**shows**  $of\text{-weak-ranking } xs \ x \ y \longleftrightarrow x \in \bigcup (set \ xs) \wedge y \in \bigcup (set \ xs) \wedge$   
 $find\text{-index } ((\in) \ x) \ xs \geq find\text{-index } ((\in) \ y) \ xs$   
 ⟨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  $\Rightarrow$  '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*:

$x \succeq [of\text{-ranking } xs] \ y \longleftrightarrow x \succeq [of\text{-weak-ranking } (map \ (\lambda x. \ \{x\}) \ xs)] \ y$

⟨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*  $[] = (\lambda \_ \ . \ False)$

⟨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**  $a \succ [of\text{-ranking } (xs \ @ \ ys)] \ b \longleftrightarrow$

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

**lemma** *not-strongly-preferred-of-ranking-iff*:

**assumes**  $a \in \text{set } xs \ b \in \text{set } xs$

**shows**  $\neg a \prec[\text{of-ranking } xs] b \longleftrightarrow a \succeq[\text{of-ranking } xs] b$

⟨proof⟩

**lemma** *of-ranking-refl*:

**assumes**  $x \in \text{set } xs$

**shows**  $x \preceq[\text{of-ranking } xs] x$

⟨proof⟩

**lemma** *of-ranking-altdef*:

**assumes** *distinct*  $xs \ x \in \text{set } xs \ y \in \text{set } xs$

**shows**  $\text{of-ranking } xs \ x \ y \longleftrightarrow \text{index } xs \ x \geq \text{index } xs \ y$

⟨proof⟩

**lemma** *of-ranking-altdef'*:

**assumes** *distinct*  $xs$

**shows**  $\text{of-ranking } xs \ x \ y \longleftrightarrow x \in \text{set } xs \wedge y \in \text{set } xs \wedge \text{index } xs \ x \geq \text{index } xs \ y$

⟨proof⟩

**lemma** *of-ranking-nth-iff*:

**assumes** *distinct*  $xs \ i < \text{length } xs \ j < \text{length } xs$

**shows**  $\text{of-ranking } xs \ (xs ! i) \ (xs ! j) \longleftrightarrow i \geq j$

⟨proof⟩

**lemma** *strongly-preferred-of-ranking-nth-iff*:

**assumes** *distinct*  $xs \ i < \text{length } xs \ j < \text{length } xs$

**shows**  $xs ! i \succ[\text{of-ranking } xs] xs ! j \longleftrightarrow i < j$

⟨proof⟩

**lemma** *of-ranking-total*:  $x \in \text{set } xs \implies y \in \text{set } xs \implies \text{of-ranking } xs \ x \ y \vee \text{of-ranking } xs \ y \ x$

⟨proof⟩

**lemma** *of-ranking-antisym*:

$x \in \text{set } xs \implies y \in \text{set } xs \implies \text{of-ranking } xs \ x \ y \implies \text{of-ranking } xs \ y \ x \implies \text{distinct } xs \implies x = y$

⟨proof⟩

**lemma** *finite-linorder-of-ranking*:

**assumes**  $\text{set } xs = A \ \text{distinct } xs$

**shows** *finite-linorder-on*  $A \ (\text{of-ranking } xs)$

⟨proof⟩

**lemma** *linorder-of-ranking*:

**assumes**  $\text{set } xs = A \ \text{distinct } xs$

**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 ( $\lambda x. f - \{x\}$ ) 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  $\longleftrightarrow$  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**  *$x \prec_{[of-ranking xs]} y \longleftrightarrow x \in set xs \wedge y \in set xs \wedge index xs x > index xs y$*   
*<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  $\implies$  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-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-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-sorts*  $\{\}$   $R = \{\{\}\}$

<proof>

**lemma** *topo-sorts-infinite*: infinite  $A \Longrightarrow$  *topo-sorts*  $A R = \{\}$

<proof>

**lemma** *topo-sorts-rec*:

finite  $A \Longrightarrow A \neq \{\} \Longrightarrow$   
*topo-sorts*  $A R = (\bigcup_{x \in \{x \in A. \forall z \in A. R x z \longrightarrow z = x\}} (\lambda xs. x \# xs)$  ' *topo-sorts*  $(A - \{x\})$   $(\lambda y z. R y z \wedge y \neq x \wedge z \neq x)$ )  
 <proof>

**lemma** *topo-sorts-cong* [*cong*]:

**assumes**  $A = B \wedge x y. x \in A \Longrightarrow y \in B \Longrightarrow x \neq y \Longrightarrow R x y = R' x y$   
**shows** *topo-sorts*  $A R =$  *topo-sorts*  $B R'$

<proof>

**lemma** *topo-sorts-correct*:

**assumes**  $\wedge x y. R x y \Longrightarrow x \in A \wedge y \in A$   
**shows** *topo-sorts*  $A R = \{xs \in \text{permutations-of-set } A. R \leq \text{of-ranking } xs\}$   
 <proof>

**lemma** *topo-sorts-nonempty*:

**assumes** finite  $A \wedge x y. R x y \Longrightarrow x \in A \wedge y \in A \wedge x y. R x y \Longrightarrow \neg R y x \text{ transp } R$   
**shows** *topo-sorts*  $A R \neq \{\}$

<proof>

**lemma** *bij-betw-topo-sorts-linorders-on*:

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

<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))
    (λx. map ((#) x) (topo-sorts-aux
      (map (map-prod id (Set.filter (λy. 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))
    (λx. map ((#) x) (topo-sorts-aux
      (map (map-prod id (Set.filter (λy. 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)))
    (λx. map ((#) x) (topo-sorts-aux
      (map (map-prod id (Set.filter (λy. y ≠ x))) (filter (λ(y,-). y ≠ x) (y#xs))))))
  ⟨proof⟩

```

**lemma** *set-topo-sorts-aux*:

```

assumes distinct (map fst xs)
assumes  $\bigwedge x ys. (x, ys) \in \text{set } xs \implies ys \subseteq \text{set } (\text{map fst } xs) - \{x\}$ 
shows  $\text{set } (\text{topo-sorts-aux } xs) =$ 
   $\text{topo-sorts } (\text{set } (\text{map fst } xs)) (\lambda x y. \exists ys. (x, ys) \in \text{set } xs \wedge y \in ys)$ 
  ⟨proof⟩

```

**lemma** *topo-sorts-code* [*code*]:

```

topo-sorts (set xs) R = (let xs' = remdups xs in
  set (topo-sorts-aux (map (λx. (x, set (filter (λy. y ≠ x ∧ R x y) xs')) xs'))))
  ⟨proof⟩

```

**end**