Linear orders as rankings

Manuel Eberl

February 10, 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*
   by (*induction xs*) *auto*

lemma *map-index-self*:
   assumes *distinct xs*
   shows   *map (index xs) xs = [0..<length xs]*
proof −
   have *xs = map ($\lambda i.\ xs\ !\ i$) [0..<length xs]*
     by (*simp add*: *map-nth*)
   also have *map (index xs) … = map id [0..<length xs]*
     unfolding *map-map* **by** (*intro map-cong*) (*use assms* **in** ‹*simp-all add: index-nth-id*›)
   finally show *?thesis*
     by *simp*
qed

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)*
   using *assms* **unfolding** *bij-betw-def* **by** (*auto simp*: *inj-on-def*)

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

lemma *is-weak-ranking-map-singleton-iff* [*simp*]:
   is-weak-ranking (map ($\lambda x.\ \{x\}$) xs) $\longleftrightarrow$ distinct xs
   by (*induction xs*) (*auto simp*: *is-weak-ranking-Cons*)

lemma *is-finite-weak-ranking-map-singleton-iff* [*simp*]:
  is-finite-weak-ranking (*map* ($\lambda x.\ \{x\}$) *xs*) $\longleftrightarrow$ *distinct xs*
  by (*induction xs*) (*auto simp*: *is-finite-weak-ranking-Cons*)

lemma *of-weak-ranking-altdef′*:
  assumes *is-weak-ranking xs*
  shows   *of-weak-ranking xs x y* $\longleftrightarrow$ $x \in \bigcup$ (*set xs*) $\land$ $y \in \bigcup$ (*set xs*) $\land$
            find-index (($\in$) *x*) *xs* $\geq$ *find-index* (($\in$) *y*) *xs*
proof (*cases* $x \in \bigcup$ (*set xs*) $\land$ $y \in \bigcup$ (*set xs*))
  case *True*
  thus *?thesis*
    using *True of-weak-ranking-altdef* [*OF assms, of x y*] **by** *auto*
next
  case *False*
  interpret *total-preorder-on* $\bigcup$ (*set xs*) *of-weak-ranking xs*
    by (*rule total-preorder-of-weak-ranking*) (*use assms* **in** *auto*)
  have $\neg$*of-weak-ranking xs x y*
    using *not-outside False* **by** *blast*
  thus *?thesis* **using** *False*
    by *blast*
qed

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**
  $i \leq j \implies i < length\ xs \implies j < length\ xs \implies\ xs\ !\ i \succeq$[*of-ranking xs*] *xs ! j*

lemma *of-ranking-conv-of-weak-ranking*:
  $x \succeq$[*of-ranking xs*] $y \longleftrightarrow x \succeq$[*of-weak-ranking* (*map* ($\lambda x.\ \{x\}$) *xs*)] y
  unfolding *of-ranking.simps of-weak-ranking.simps* **by** *fastforce*

lemma *of-ranking-imp-in-set*:
  assumes *of-ranking xs a b*
  shows $a \in set\ xs$ $b \in set\ xs$
  using *assms* **by** (*fastforce elim!*: *of-ranking.cases*)+

lemma *of-ranking-Nil* [*simp*]: *of-ranking* [] = (λ- -. *False*)
  by (*auto simp*: *of-ranking.simps fun-eq-iff*)

lemma *of-ranking-Nil′* [*code*]: *of-ranking* [] *x y* = *False*
  by *simp*

lemma *of-ranking-Cons* [*code*]:
  $x \succeq$[*of-ranking* (*z#zs*)] $y \longleftrightarrow x = z \land y \in set$ (*z#zs*) $\lor x \succeq$[*of-ranking zs*] y
  by (*auto simp*: *of-ranking-conv-of-weak-ranking of-weak-ranking-Cons*)

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-ranking* xs a b)
  using *assms of-ranking-imp-in-set*[*of* xs a b] **by** (*auto simp*: *of-ranking-Cons*)

lemma *of-ranking-append*:
  $x \succeq$[*of-ranking* (xs @ ys)] y $\longleftrightarrow$ $x \in set$ xs $\wedge$ $y \in set$ ys $\vee$ $x \succeq$[*of-ranking* xs] y $\vee$ $x \succeq$[*of-ranking*
ys] y
  by (*induction* xs) (*auto simp*: *of-ranking-Cons*)

lemma *of-ranking-strongly-preferred-Cons-iff*:
  assumes *distinct* (x # xs)
  shows $a \succ$[*of-ranking* (x # xs)] b $\longleftrightarrow$ $x = a$ $\wedge$ $b \in set$ xs $\vee$ $a \succ$[*of-ranking* xs] b
  using *assms of-ranking-imp-in-set*[*of* xs]
  by (*auto simp*: *strongly-preferred-def of-ranking-Cons*)

lemma *of-ranking-strongly-preferred-append-iff*:
  assumes *distinct* (xs @ ys)
  shows $a \succ$[*of-ranking* (xs @ ys)] b $\longleftrightarrow$
            $a \in set$ xs $\wedge$ $b \in set$ ys $\vee$ $a \succ$[*of-ranking* xs] b $\vee$ $a \succ$[*of-ranking* ys] b
  using *assms of-ranking-imp-in-set*[*of* xs a b] *of-ranking-imp-in-set*[*of* ys a b]
          of-ranking-imp-in-set[*of* xs b a] *of-ranking-imp-in-set*[*of* ys b a]
  unfolding *strongly-preferred-def of-ranking-append distinct-append set-eq-iff Int-iff empty-iff*
  by *metis*

lemma *not-strongly-preferred-of-ranking-iff*:
  assumes $a \in set$ xs $b \in set$ xs
  shows $\neg a \prec$[*of-ranking* xs] b $\longleftrightarrow$ $a \succeq$[*of-ranking* xs] b
  using *assms* **unfolding** *strongly-preferred-def*
  by (*metis index-less-size-conv linorder-le-cases nth-index of-ranking.intros*)

lemma *of-ranking-refl*:
  assumes $x \in set$ xs
  shows $x \preceq$[*of-ranking* xs] x
  using *assms* **by** (*induction* xs) (*auto simp*: *of-ranking-Cons*)

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
  unfolding *of-ranking-conv-of-weak-ranking*
  by (*subst of-weak-ranking-altdef*)
    (*use assms* **in** ‹*auto simp*: *index-def find-index-map eq-commute*[*of* - y] *eq-commute*[*of* - x]›)

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
  unfolding *of-ranking-conv-of-weak-ranking*
  by (*subst of-weak-ranking-altdef'*)

(*use assms* **in** ⟨*auto simp: index-def find-index-map eq-commute*[*of - y*] *eq-commute*[*of - x*]⟩)

lemma *of-ranking-nth-iff*:
  assumes *distinct xs i < length xs j < length xs*
  shows   *of-ranking xs* (*xs ! i*) (*xs ! j*) ⟷ *i ≥ j*
  using *assms* **by** (*simp add: index-nth-id of-ranking-altdef*)

lemma *strongly-preferred-of-ranking-nth-iff*:
  assumes *distinct xs i < length xs j < length xs*
  shows   *xs ! i* ≻[*of-ranking xs*] *xs ! j* ⟷ *i < j*
  using *assms* **by** (*auto simp: strongly-preferred-def of-ranking-nth-iff*)

lemma *of-ranking-total*: *x* ∈ *set xs* ⟹ *y* ∈ *set xs* ⟹ *of-ranking xs x y* ∨ *of-ranking xs y x*
  by (*induction xs*) (*auto simp: of-ranking-Cons*)

lemma *of-ranking-antisym*:
  x ∈ *set xs* ⟹ *y* ∈ *set xs* ⟹ *of-ranking xs x y* ⟹ *of-ranking xs y x* ⟹ *distinct xs* ⟹ *x* =
y
  by (*simp add: of-ranking-altdef′*)

lemma *finite-linorder-of-ranking*:
  assumes *set xs = A distinct xs*
  shows   *finite-linorder-on A* (*of-ranking xs*)
proof −
  interpret *total-preorder-on A of-ranking xs*
    unfolding *of-ranking-conv-of-weak-ranking*
    by (*rule total-preorder-of-weak-ranking*) (*use assms* **in** *auto*)
  show *?thesis*
  proof
    fix *x y* **assume** *of-ranking xs x y of-ranking xs y x*
    thus *x = y*
      by (*metis assms*(*1,2*) *index-eq-index-conv nle-le not-outside*(*2*) *of-ranking-altdef*)
  qed (*use assms*(*1*) **in** *auto*)
qed

lemma *linorder-of-ranking*:
  assumes *set xs = A distinct xs*
  shows   *linorder-on A* (*of-ranking xs*)
proof −
  interpret *finite-linorder-on A of-ranking xs*
    by (*rule finite-linorder-of-ranking*) *fact+*
  show *?thesis* **..**
qed

lemma *total-preorder-of-ranking*:
  assumes *set xs = A distinct xs*
  shows   *total-preorder-on A* (*of-ranking xs*)
  unfolding *of-ranking-conv-of-weak-ranking*

5

by (*rule total-preorder-of-weak-ranking*) (*use assms* **in** *auto*)

1.3 Transformations

lemma *map-relation-of-ranking*:
  map-relation f (*of-ranking xs*) = *of-weak-ranking* (*map* ($\lambda x.\ f\ -\ `\ \{x\}$) *xs*)
  unfolding *of-ranking-conv-of-weak-ranking of-weak-ranking-map map-map o-def* **..**

lemma *of-ranking-map*: *of-ranking* (*map f xs*) = *comap-relation f* (*of-ranking xs*)
  by (*induction xs*) (*auto simp*: *comap-relation-def of-ranking-Cons fun-eq-iff*)

lemma *of-ranking-permute′*:
  assumes *f permutes set xs*
  shows   *map-relation f* (*of-ranking xs*) = *of-ranking* (*map* (*inv f*) *xs*)
  unfolding *of-ranking-conv-of-weak-ranking*
  by (*subst of-weak-ranking-permute′*) (*use assms* **in** ‹*auto simp*: *map-map o-def*›)

lemma *of-ranking-permute*:
  assumes *f permutes set xs*
  shows   *of-ranking* (*map f xs*) = *map-relation* (*inv f*) (*of-ranking xs*)
  using *of-ranking-permute′*[*OF permutes-inv*[*OF assms*]] *assms*
  by (*simp add*: *inv-inv-eq permutes-bij*)

lemma *of-ranking-rev* [*simp*]:
  of-ranking (*rev xs*) *x y* ⟷ *of-ranking xs y x*
  unfolding *of-ranking-conv-of-weak-ranking* **by** (*simp flip*: *rev-map*)

lemma *of-ranking-filter*:
  of-ranking (*filter P xs*) = *restrict-relation* $\{x.\ P\ x\}$ (*of-ranking xs*)
  by (*induction xs*) (*auto simp*: *of-ranking-Cons restrict-relation-def fun-eq-iff*)

lemma *strongly-preferred-of-ranking-conv-index*:
  assumes *distinct xs*
  shows $x \prec$[*of-ranking xs*] y ⟷ $x \in set\ xs \land y \in set\ xs \land index\ xs\ x > index\ xs\ y$
  unfolding *strongly-preferred-def* **using** *of-ranking-altdef′*[*OF assms*] **by** *auto*

lemma *restrict-relation-of-weak-ranking-Cons*:
  assumes *distinct* (*x* # *xs*)
  shows   *restrict-relation* (*set xs*) (*of-ranking* (*x* # *xs*)) = *of-ranking xs*
proof −
  from *assms* **interpret** *R*: *total-preorder-on set xs of-ranking xs*
    by (*intro total-preorder-of-ranking*) *auto*
  from *assms* **show** *?thesis* **using** *R.not-outside*
    by (*intro ext*) (*auto simp*: *restrict-relation-def of-ranking-Cons*)
qed

lemma *of-ranking-zero-upt-nat*:
  of-ranking [0::*nat*..<*n*] = ($\lambda x\ y.\ x \geq y \land x < n$)
  by (*induction n*) (*auto simp*: *of-ranking-append of-ranking-Cons fun-eq-iff*)

lemma *of-ranking-rev-zero-upt-nat*:
  of-ranking (*rev* [*0::nat..<n*]) = (λ*x y. x ≤ y ∧ y < n*)
  by (*induction n*) (*auto simp*: *of-ranking-Cons fun-eq-iff*)

lemma *sorted-wrt-ranking*: *distinct xs* ⟹ *sorted-wrt* (*of-ranking xs*) (*rev xs*)
  unfolding *sorted-wrt-iff-nth-less* **by** (*force simp*: *of-ranking.simps rev-nth*)

1.4 Inverse operation and isomorphism

lemma (**in** *finite-linorder-on*) *of-ranking-ranking*: *of-ranking* (*ranking le*) = *le*
proof −
  have *of-ranking* (*ranking le*) =
        of-weak-ranking (*map* (λ*x.* {*the-elem x*}) (*weak-ranking le*))
    unfolding *of-ranking-conv-of-weak-ranking ranking-def* **by** (*simp add*: *map-map o-def*)
  also have *map* (λ*x.* {*the-elem x*}) (*weak-ranking le*) = *map* (λ*x. x*) (*weak-ranking le*)
    by (*intro map-cong HOL.refl*)
       (*metis is-singleton-the-elem singleton-weak-ranking*)+
  also have *of-weak-ranking* (*map* (λ*x. x*) (*weak-ranking le*)) = *le*
    using *of-weak-ranking-weak-ranking*[*OF finite-total-preorder-on-axioms*] **by** *simp*
  finally show *?thesis* .
qed

lemma (**in** *finite-linorder-on*) *distinct-ranking*: *distinct* (*ranking le*)
  using *weak-ranking-ranking weak-ranking-total-preorder*(*1*) **by** *simp*

lemma *ranking-of-ranking*:
  assumes *distinct xs*
  shows   *ranking* (*of-ranking xs*) = *xs*
proof −
  have *ranking* (*of-ranking xs*) = *map the-elem* (*weak-ranking* (*of-weak-ranking* (*map* (λ*x.* {*x*})
xs)))
    unfolding *ranking-def of-ranking-conv-of-weak-ranking* ..
  also have . . . = *xs*
    by (*subst weak-ranking-of-weak-ranking*) (*use assms* **in** ‹*auto simp*: *o-def*›)
  finally show *?thesis* .
qed

lemma (**in** *finite-linorder-on*) *set-ranking*: *set* (*ranking le*) = *carrier*
  using *weak-ranking-Union weak-ranking-ranking* **by** *auto*

lemma *bij-betw-permutations-of-set-finite-linorders-on*:
  bij-betw of-ranking (*permutations-of-set A*) {*R. finite-linorder-on A R*}
  by (*rule bij-betwI*[*of - - - ranking*])
     (*auto simp*: *finite-linorder-on.of-ranking-ranking ranking-of-ranking*
             permutations-of-set-def finite-linorder-on.distinct-ranking
             finite-linorder-on.set-ranking intro: *finite-linorder-of-ranking*)

lemma *bij-betw-permutations-of-set-finite-linorders-on′*:

bij-betw ranking {R. finite-linorder-on A R} (permutations-of-set A)
by (*rule bij-betwI[of - - - of-ranking]*)
  (*auto simp*: *finite-linorder-on.of-ranking-ranking ranking-of-ranking*
          permutations-of-set-def finite-linorder-on.distinct-ranking
          finite-linorder-on.set-ranking intro: *finite-linorder-of-ranking*)

lemma *card-linorders-on*:
  assumes *finite A*
  shows   *card {R. linorder-on A R} = fact (card A)*
proof −
  have *{R. linorder-on A R} = {R. finite-linorder-on A R}*
    using *assms* **by** (*simp add*: *finite-linorder-on-def finite-linorder-on-axioms-def*)
  also have *card* ... *= card (permutations-of-set A)*
    using *bij-betw-same-card[OF bij-betw-permutations-of-set-finite-linorders-on[of A]]* **by** *simp*
  also have ... *= fact (card A)*
    using *assms* **by** *simp*
  finally show *?thesis* .
qed

lemma *finite-linorders-on* [*intro*]:
  assumes *finite A*
  shows   *finite {R. linorder-on A R}*
proof −
  from *assms* **have** *finite (permutations-of-set A)*
    by *simp*
  also have *finite (permutations-of-set A) ⟷ finite {R. finite-linorder-on A R}*
    by (*rule bij-betw-finite[OF bij-betw-permutations-of-set-finite-linorders-on]*)
  also have *{R. finite-linorder-on A R} = {R. linorder-on A R}*
    using *assms* **by** (*simp add*: *finite-linorder-on-axioms.intro finite-linorder-on-def*)
  finally show *?thesis* .
qed

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*

$\bigcup x \in \{x \in A.\ \forall z \in A.\ R\ x\ z \longrightarrow z = x\}.\ (\lambda xs.\ x\ \#\ xs)\ `\ \textit{topo-sorts}\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x))$
by *auto*
termination
proof (*relation Wellfounded.measure* (*card* ∘ *fst*), *goal-cases*)
  case (*2 A R x*)
  show *?case*
  proof (*cases infinite $A \vee A = \{\}$*)
    case *False*
    have $A - \{x\} \subset A$
      using *2* **by** *auto*
    with *False* **have** $card\ (A - \{x\}) < card\ A$
      by (*intro psubset-card-mono*) *auto*
    thus *?thesis*
      using *False 2* **by** *simp*
  qed (*use 2* **in** *auto*)
qed *auto*

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

lemma *topo-sorts-empty* [*simp*]: *topo-sorts* $\{\}\ R = \{[]\}$
  by (*subst topo-sorts.simps*) *auto*

lemma *topo-sorts-infinite*: *infinite $A \Longrightarrow$ topo-sorts $A\ R = \{\}$*
  by (*subst topo-sorts.simps*) *auto*

lemma *topo-sorts-rec*:
  finite $A \Longrightarrow A \neq \{\} \Longrightarrow$
    topo-sorts $A\ R = (\bigcup x \in \{x \in A.\ \forall z \in A.\ R\ x\ z \longrightarrow z = x\}.$
    $(\lambda xs.\ x\ \#\ xs)\ `\ topo-sorts\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x))$
  by (*subst topo-sorts.simps*) *simp-all*

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-sorts $A\ R =$ topo-sorts $B\ R'$*
proof (*cases finite A*)
  case *True*
  from *this* **and** *assms(2)* **show** *?thesis*
    unfolding *assms(1)*[*symmetric*]
  proof (*induction arbitrary: $R\ R'$ rule: finite-psubset-induct*)
    case (*psubset A R R'*)
    show *?case*
    proof (*cases $A = \{\}$*)
      case *False*
      have $(\bigcup x \in \{x \in A.\ \forall z \in A.\ R\ x\ z \longrightarrow z = x\}.\ (\#)\ x\ `\ topo-sorts\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x)) =$
        $(\bigcup x \in \{x \in A.\ \forall z \in A.\ R'\ x\ z \longrightarrow z = x\}.\ (\#)\ x\ `\ topo-sorts\ (A - \{x\})\ (\lambda y\ z.\ R'\ y\ z \wedge y \neq x \wedge z \neq x))$
        using *psubset.prems psubset.hyps*

9

by (*intro arg-cong*[*of* - - $\bigcup$] *image-cong refl psubset.IH*) *auto*
    thus *?thesis*
    by (*subst* (*1 2*) *topo-sorts-rec*) (*use False psubset.hyps* **in** *simp-all*)
  qed *auto*
qed
qed (*simp-all add*: *assms*(*1*) *topo-sorts-infinite*)

lemma *topo-sorts-correct*:
  assumes $\bigwedge x\ y.\ R\ x\ y \implies x \in A \wedge y \in A$
  shows   *topo-sorts A R = {xs∈permutations-of-set A. R ≤ of-ranking xs}*
  using *assms*
proof (*induction A R rule*: *topo-sorts.induct*)
  case (*1 A R*)
  note *R = 1.prems*

  show *?case*
  proof (*cases A = {}* $\vee$ *infinite A*)
    case *True*
    thus *?thesis* **using** *R*
      by (*auto simp*: *topo-sorts-infinite permutations-of-set-infinite*)
  next
    case *False*
    define *M* **where** *M = {x∈A. ∀ z∈A. R x z* $\longrightarrow$ *z = x}*
    define R' **where** $R' = (\lambda x\ y\ z.\ R\ y\ z \wedge y \neq x \wedge z \neq x)$

    have *IH*: *topo-sorts* $(A - \{x\})\ (R'\ x) = \{xs \in$ *permutations-of-set* $(A - \{x\}).\ (R'\ x) \leq$ *of-ranking xs}*
      if *x*: $x \in M$ **for** *x*
      unfolding R'-*def* **by** (*rule 1.IH*) (*use False x R* **in** ‹*auto simp*: *M-def*›)

    have *{xs∈permutations-of-set A. R ≤ of-ranking xs} =*
        $(\bigcup x \in A.\ ((\#)\ x)$ ' $\{xs \in$ *permutations-of-set* $(A - \{x\}).\ R \leq$ *of-ranking* $(x \# xs)\})$
    by (*subst permutations-of-set-nonempty*) (*use False* **in** *auto*)
    also have . . . $= (\bigcup x \in A.\ ((\#)\ x)$ ' $\{xs \in$ *permutations-of-set* $(A - \{x\}).\ x \in M \wedge R'\ x \leq$ *of-ranking xs}*$)$
    proof (*intro arg-cong*[*of* - - $\bigcup$] *image-cong Collect-cong conj-cong refl*)
      fix *x xs*
      assume *x*: $x \in A$ **and** *xs*: $xs \in$ *permutations-of-set* $(A - \{x\})$
      from *xs* **have** *xs'*: *set xs =* $A - \{x\}$ *distinct xs*
        by (*auto simp*: *permutations-of-set-def*)

      have *R ≤ of-ranking* $(x \# xs) \longleftrightarrow (\forall y\ z.\ R\ y\ z \longrightarrow z = x \wedge y \in$ *set* $(x \# xs) \vee$ *of-ranking xs y z*$)$
        unfolding *le-fun-def of-ranking-Cons* **by** *auto*
      also have $(\lambda y\ z.\ R\ y\ z \longrightarrow z = x \wedge y \in$ *set* $(x \# xs) \vee$ *of-ranking xs y z*$) =$
          $(\lambda y\ z.\ R\ y\ z \longrightarrow ((y = x \longrightarrow z = x) \wedge (y \neq x \wedge z \neq x \longrightarrow$ *of-ranking xs y z*$)))$
        unfolding *fun-eq-iff* **using** *R of-ranking-altdef' xs'*(*1,2*) **by** *fastforce*
      also have $(\forall y\ z.\ \dots\ y\ z) \longleftrightarrow (\forall z.\ R\ x\ z \longrightarrow z = x) \wedge R'\ x \leq$ *of-ranking xs*
        unfolding *le-fun-def of-ranking-Cons* R'-*def* **by** *auto*

also have $(\forall\, z.\ R\ x\ z \longrightarrow z = x) \longleftrightarrow x \in M$
  unfolding *M-def* **using** *x R* **by** *auto*
finally show $(R \le \textit{of-ranking}\ (x\ \#\ xs)) = (x \in M \wedge R'\ x \le \textit{of-ranking}\ xs)$ **.**
qed
also have $\ldots\ = (\bigcup x{\in}M.\ ((\#)\ x)\ {}^{\backprime} \{xs{\in}\textit{permutations-of-set}\ (A - \{x\}).\ R'\ x \le \textit{of-ranking}$
$xs\})$
  unfolding *M-def* **by** *blast*
also have $\ldots\ = (\bigcup x{\in}M.\ ((\#)\ x)\ {}^{\backprime}\ \textit{topo-sorts}\ (A - \{x\})\ (R'\ x))$
  using *IH* **by** *blast*
also have $\ldots\ = \textit{topo-sorts}\ A\ R$
  unfolding *R'-def M-def* **using** *False* **by** (*subst (2) topo-sorts-rec*) *simp-all*
finally show *?thesis* **..**
  qed
qed

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-sorts* $A\ R \ne \{\}$
  using *assms*
proof (*induction A R rule*: *topo-sorts.induct*)
  case (*1 A R*)
  define R' **where** $R' = (\lambda x\ y.\ x \in A \wedge y \in A \wedge x = y \vee R\ x\ y)$
  interpret R': *order-on* $A\ R'$
    by *standard* (*use 1.prems(2,3)* **in** ‹*auto simp*: *R'-def intro*: *transpD[OF* ‹*transp R*›*]*›)

  show *?case*
  proof (*cases* $A = \{\}$)
    case *False*
    define M **where** $M = \textit{Max-wrt-among}\ R'\ A$
    have $M \ne \{\}$
      unfolding *M-def* **by** (*rule R'.Max-wrt-among-nonempty*) (*use False* ‹*finite A*› **in** *simp-all*)
    obtain x **where** x: $x \in M$
      using ‹$M \ne \{\}$› **by** *blast*
    have *M-altdef*: $M = \{x{\in}A.\ \forall\, z{\in}A.\ R\ x\ z \longrightarrow z = x\}$
      unfolding *M-def Max-wrt-among-def R'-def* **using** *1.prems* **by** *blast*

    define L **where** $L = \textit{topo-sorts}\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \ne x \wedge z \ne x)$
    have $L \ne \{\}$
      unfolding *L-def*
    proof (*rule 1.IH*)
      show *transp* $(\lambda a\ b.\ R\ a\ b \wedge a \ne x \wedge b \ne x)$
        using ‹*transp R*› **unfolding** *transp-def* **by** *blast*
    qed (*use 1.prems(2,3) False x* ‹*finite A*› **in** ‹*auto simp*: *M-altdef*›)

    have *topo-sorts* $A\ R =$
        $(\bigcup x{\in}\{x{\in}A.\ \forall\, z{\in}A.\ R\ x\ z \longrightarrow z = x\}.$
        $(\lambda xs.\ x\ \#\ xs)\ {}^{\backprime}\ \textit{topo-sorts}\ (A - \{x\})\ (\lambda y\ z.\ R\ y\ z \wedge y \ne x \wedge z \ne x))$
      by (*subst topo-sorts.simps*) (*use False* ‹*finite A*› **in** *simp-all*)
    also have $\{x{\in}A.\ \forall\, z{\in}A.\ R\ x\ z \longrightarrow z = x\} = M$

11

      unfolding *M-altdef* **..**
    finally show *topo-sorts A R* $\neq$ *{}*
      using ‹*L* $\neq$ *{}*› ‹*x* $\in$ *M*› **unfolding** *L-def* **by** *blast*
  qed *auto*
qed

lemma *bij-betw-topo-sorts-linorders-on*:
  assumes $\bigwedge x\ y.\ R\ x\ y \implies x \in A \wedge y \in A$
  shows   *bij-betw of-ranking* (*topo-sorts A R*) *{R′. finite-linorder-on A R′* $\wedge$ *R* $\leq$ *R′}*
proof $-$
  have *bij-betw of-ranking {xs∈permutations-of-set A. R* $\leq$ *of-ranking xs}*
      {R′∈{R′. finite-linorder-on A R′}. R $\leq$ *R′}*
    using *bij-betw-permutations-of-set-finite-linorders-on*
    by (*rule bij-betw-Collect*) *auto*
  also have *{xs∈permutations-of-set A. R* $\leq$ *of-ranking xs} = topo-sorts A R*
    by (*subst topo-sorts-correct*) (*use assms* **in** *auto*)
  finally show *?thesis*
    by *simp*
qed

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*))
      ($\lambda x.\ map$ ((#) *x*) (*topo-sorts-aux*
        (*map* (*map-prod id* (*Set.filter* ($\lambda y.\ y \neq x$))) (*filter* (λ(*y*,-). $y \neq x$) *xs*))))))
  by *auto*
termination
  by (*relation Wellfounded.measure length*)
    (*auto simp*: *length-filter-less*)

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

lemma *topo-sorts-aux-Nil* [*simp*]: *topo-sorts-aux [] = [[]]*
  by (*subst topo-sorts-aux.simps*) *auto*

lemma *topo-sorts-aux-rec*:
  xs $\neq$ *[]* $\implies$ *topo-sorts-aux xs =*
    List.bind (*map fst* (*filter* (λ(-,*ys*). *ys = {}*) *xs*))
      ($\lambda x.\ map$ ((#) *x*) (*topo-sorts-aux*
        (*map* (*map-prod id* (*Set.filter* ($\lambda y.\ y \neq x$))) (*filter* (λ(*y*,-). $y \neq x$) *xs*))))
  by (*subst topo-sorts-aux.simps*) *auto*

lemma *topo-sorts-aux-Cons*:
  topo-sorts-aux (*y*#*xs*) =
    List.bind (*map fst* (*filter* (λ(-,*ys*). *ys = {}*) (*y*#*xs*)))

$(\lambda x.\ map\ ((\#)\ x)\ (topo\text{-}sorts\text{-}aux$
$\qquad (map\ (map\text{-}prod\ id\ (Set.filter\ (\lambda y.\ y \neq x)))\ (filter\ (\lambda(y,\text{-}).\ y \neq x)\ (y\#xs))))))$
by (*rule topo-sorts-aux-rec*) *auto*

lemma *set-topo-sorts-aux*:
  assumes *distinct* (*map fst xs*)
  assumes $\bigwedge x\ ys.\ (x,\ ys) \in set\ xs \Longrightarrow ys \subseteq set\ (map\ fst\ xs) - \{x\}$
  shows   *set* (*topo-sorts-aux xs*) $=$
         topo-sorts (*set* (*map fst xs*)) $(\lambda x\ y.\ \exists ys.\ (x,\ ys) \in set\ xs \wedge y \in ys)$
  using *assms*
proof (*induction xs rule*: *topo-sorts-aux.induct*)
  case (*1 xs*)
  show *?case*
  proof (*cases xs* $=$ [])
    case *True*
    thus *?thesis*
      by (*simp add*: *topo-sorts.simps*[*of* {}] *topo-sorts-aux.simps*[*of* []])
  next
    case *False*
    define *M* **where** $M = set\ (map\ fst\ (filter\ (\lambda(\text{-},ys).\ ys = \{\})\ xs))$
    define *xs′* **where** $xs' = (\lambda x.\ map\ (map\text{-}prod\ id\ (Set.filter\ (\lambda y.\ y \neq x)))\ (filter\ (\lambda(y,\text{-}).\ y \neq x)\ xs))$
    define *R′* **where** $R' = (\lambda x\ a\ b.\ \exists ys.\ (a,\ ys) \in set\ (xs'\ x) \wedge b \in ys)$

    have *IH*: *set* (*topo-sorts-aux* (*xs′ x*)) $=$ *topo-sorts* (*set* (*map fst* (*xs′ x*))) (*R′ x*)
      if $x \in M$ **for** *x*
      unfolding *xs′-def R′-def*
    proof (*rule 1.IH*, *goal-cases*)
      case *2*
      show *?case* **using** *that* **by** (*auto simp*: *M-def*)
    next
      case *3*
      thus *?case* **using** *1.prems*
        by (*auto intro*!: *distinct-filter simp*: *distinct-map intro*: *inj-on-subset*)
    next
      case *4*
      thus *?case* **using** *1.prems* **by** *fastforce*
    qed *fact+*

    have *topo-sorts* (*set* (*map fst xs*)) $(\lambda x\ y.\ \exists ys.\ (x,\ ys) \in set\ xs \wedge y \in ys) =$
        $(\bigcup x \in \{x \in set\ (map\ fst\ xs).\ \forall z \in set\ (map\ fst\ xs).\ (\exists ys.\ (x,\ ys) \in set\ xs \wedge z \in ys) \longrightarrow z = x\}.$
           $(\#)\ x\ '\ topo\text{-}sorts\ (set\ (map\ fst\ xs) - \{x\})\ (\lambda y\ z.\ (\exists ys.\ (y,\ ys) \in set\ xs \wedge z \in ys) \wedge y \neq x \wedge z \neq x))$
      by (*subst topo-sorts-rec*) (*use False* **in** *simp-all*)
    also have $\{x \in set\ (map\ fst\ xs).\ \forall z \in set\ (map\ fst\ xs).\ (\exists ys.\ (x,\ ys) \in set\ xs \wedge z \in ys) \longrightarrow z = x\} = M$
      (**is** *?lhs* $=$ *?rhs*)
    proof (*intro equalityI subsetI*)

13

fix x **assume** $x \in \textit{?rhs}$
  thus $x \in \textit{?lhs}$
    using *1*.*prems* **by** (*fastforce simp*: *M-def distinct-map inj-on-def*)
  next
  fix x **assume** $x \in \textit{?lhs}$
   hence x: $x \in \textit{set } (\textit{map fst xs}) \bigwedge z\ ys.\ z \in \textit{set } (\textit{map fst xs}) \Longrightarrow (x,\ ys) \in \textit{set xs} \land z \in ys$
$\Longrightarrow z = x$
    by *blast+*
  from $x(1)$ **obtain** ys **where** ys: $(x,\ ys) \in \textit{set xs}$
    by *force*
  have $ys \subseteq \{\}$
  proof
    fix y **assume** $y \in ys$
    with ys **show** $y \in \{\}$
      using $x(2)[\textit{of } y\ ys]$ *1*.*prems* **by** *auto*
  qed
  thus $x \in \textit{?rhs}$
    unfolding *M-def* **using** $x(1)$ ys **by** (*auto simp*: *image-iff*)
  qed
  also have $(\lambda x.\ \textit{set } (\textit{map fst xs}) - \{x\}) = (\lambda x.\ \textit{set } (\textit{map fst } (\textit{xs}'\ x)))$
    by (*force simp*: *xs'-def fun-eq-iff*)
  also have $(\lambda x\ y\ z.\ (\exists ys.\ (y,\ ys) \in \textit{set xs} \land z \in ys) \land y \neq x \land z \neq x) = R'$
    unfolding *R'-def* **using** *1*.*prems*
    by (*auto simp*: *fun-eq-iff distinct-map inj-on-def xs'-def map-prod-def*
                   case-prod-unfold image-iff)
  also have $(\bigcup x \in M.\ (\#)\ x\ `\ \textit{topo-sorts } (\textit{set } (\textit{map fst } (\textit{xs}'\ x)))\ (R'\ x)) =$
          $(\bigcup x \in M.\ (\#)\ x\ `\ \textit{set } (\textit{topo-sorts-aux } (\textit{xs}'\ x)))$
    using *IH* **by** *blast*
  also have $\ldots = \textit{set } (\textit{topo-sorts-aux xs})$
    by (*subst* (*2*) *topo-sorts-aux-rec*) (*use False* **in** ‹*auto simp*: *M-def xs'-def List.bind-def*›)
  finally show *?thesis* **..**
 qed
qed

lemma *topo-sorts-code* [*code*]:
 topo-sorts (*set xs*) *R* = (**let** $xs' = \textit{remdups xs}$ **in**
  set (*topo-sorts-aux* (*map* ($\lambda x.\ (x,\ \textit{set } (\textit{filter } (\lambda y.\ y \neq x \land R\ x\ y)\ xs')))$) xs')))
proof −
 define xs' **where** $xs' = \textit{remdups xs}$
 have *set* (*topo-sorts-aux* (*map* ($\lambda x.\ (x,\ \textit{set } (\textit{filter } (\lambda y.\ y \neq x \land R\ x\ y)\ xs')))$) xs')) =
      topo-sorts (*set xs*) ($\lambda x\ y.\ \exists ys.\ (x,\ ys) \in (\lambda x.\ (x,\ \textit{set } (\textit{filter } (\lambda y.\ y \neq x \land R\ x\ y)\ xs')))$ `
set $xs' \land y \in ys$)
   by (*subst set-topo-sorts-aux*) (*auto simp*: *o-def xs'-def*)
 also have ($\lambda x\ y.\ \exists ys.\ (x,\ ys) \in (\lambda x.\ (x,\ \textit{set } (\textit{filter } (\lambda y.\ y \neq x \land R\ x\ y)\ xs')))$) ` *set* $xs' \land y \in$
ys) =
      ($\lambda x\ y.\ x \in \textit{set xs} \land y \in \textit{set xs} \land x \neq y \land R\ x\ y$)
   by (*auto simp*: *xs'-def image-iff*)
 also have *topo-sorts* (*set xs*) $\ldots$ = *topo-sorts* (*set xs*) *R*
   by (*rule topo-sorts-cong*) *auto*

   finally show *?thesis*
     by (*simp add*: *Let-def xs'-def*)
qed

end