Well-Quasi-Orders

Christian Sternagel*

October 13, 2025

Abstract

Based on Isabelle/HOL's type class for preorders, we introduce a type class for well-quasi-orders (wqo) which is characterized by the absence of "bad" sequences (our proofs are along the lines of the proof of Nash-Williams [1], from which we also borrow terminology). Our main results are instantiations for the product type, the list type, and a type of finite trees, which (almost) directly follow from our proofs of (1) Dickson's Lemma, (2) Higman's Lemma, and (3) Kruskal's Tree Theorem. More concretely:

1. If the sets A and B are wqo then their Cartesian product is wqo.
2. If the set A is wqo then the set of finite lists over A is wqo.
3. If the set A is wqo then the set of finite trees over A is wqo.

Contents

*The research was funded by the Austrian Science Fund (FWF): J3202.

1 Infinite Sequences

Some useful constructions on and facts about infinite sequences.

theory *Infinite-Sequences*
imports *Main*
begin

The set of all infinite sequences over elements from A.

definition $SEQ\ A = \{f{::}nat \Rightarrow 'a.\ \forall i.\ f\ i \in A\}$

lemma *SEQ-iff* [*iff*]:
$f \in SEQ\ A \longleftrightarrow (\forall i.\ f\ i \in A)$
by (*auto simp*: *SEQ-def*)

The i-th "column" of a set B of infinite sequences.

definition $\mathit{ith}\ B\ i = \{f\ i \mid f.\ f \in B\}$

lemma *ithI* [*intro*]:
$f \in B \Longrightarrow f\ i = x \Longrightarrow x \in \mathit{ith}\ B\ i$
by (*auto simp*: *ith-def*)

lemma *ithE* [*elim*]:
$[\![x \in \mathit{ith}\ B\ i;\ \bigwedge f.\ [\![f \in B;\ f\ i = x]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
by (*auto simp*: *ith-def*)

lemma *ith-conv*:
$x \in \mathit{ith}\ B\ i \longleftrightarrow (\exists f \in B.\ x = f\ i)$
by *auto*

The restriction of a set B of sequences to sequences that are equal to a given sequence f up to position i.

definition $\mathit{eq\text{-}upto} :: (\mathit{nat} \Rightarrow {'a})\ \mathit{set} \Rightarrow (\mathit{nat} \Rightarrow {'a}) \Rightarrow \mathit{nat} \Rightarrow (\mathit{nat} \Rightarrow {'a})\ \mathit{set}$
where
$\mathit{eq\text{-}upto}\ B\ f\ i = \{g \in B.\ \forall j < i.\ f\ j = g\ j\}$

lemma *eq-uptoI* [*intro*]:
$[\![g \in B;\ \bigwedge j.\ j < i \Longrightarrow f\ j = g\ j]\!] \Longrightarrow g \in \mathit{eq\text{-}upto}\ B\ f\ i$
by (*auto simp*: *eq-upto-def*)

lemma *eq-uptoE* [*elim*]:
$[\![g \in \mathit{eq\text{-}upto}\ B\ f\ i;\ [\![g \in B;\ \bigwedge j.\ j < i \Longrightarrow f\ j = g\ j]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
by (*auto simp*: *eq-upto-def*)

lemma *eq-upto-Suc*:
$[\![g \in \mathit{eq\text{-}upto}\ B\ f\ i;\ g\ i = f\ i]\!] \Longrightarrow g \in \mathit{eq\text{-}upto}\ B\ f\ (\mathit{Suc}\ i)$
by (*auto simp*: *eq-upto-def less-Suc-eq*)

lemma *eq-upto-0* [*simp*]:
$\mathit{eq\text{-}upto}\ B\ f\ 0 = B$
by (*auto simp*: *eq-upto-def*)

lemma *eq-upto-cong* [*fundef-cong*]:
assumes $\bigwedge j.\ j < i \Longrightarrow f\ j = g\ j$ **and** $B = C$
shows $\mathit{eq\text{-}upto}\ B\ f\ i = \mathit{eq\text{-}upto}\ C\ g\ i$
using *assms* **by** (*auto simp*: *eq-upto-def*)

1.1 Lexicographic Order on Infinite Sequences

definition $\mathit{LEX}\ P\ f\ g \longleftrightarrow (\exists i{::}\mathit{nat}.\ P\ (f\ i)\ (g\ i) \wedge (\forall j{<}i.\ f\ j = g\ j))$
abbreviation $\mathit{LEXEQ}\ P \equiv (\mathit{LEX}\ P)^{==}$

lemma *LEX-imp-not-LEX*:
assumes $\mathit{LEX}\ P\ f\ g$

```
    and [dest]: ⋀x y z. P x y ⟹ P y z ⟹ P x z
    and [simp]: ⋀x. ¬ P x x
  shows ¬ LEX P g f
proof −
  { fix i j :: nat
    assume P (f i) (g i) and ∀ k<i. f k = g k
      and P (g j) (f j) and ∀ k<j. g k = f k
    then have False by (cases i < j) (auto simp: not-less dest!: le-imp-less-or-eq)
}
  then show ¬ LEX P g f using ‹LEX P f g› unfolding LEX-def by blast
qed

lemma LEX-cases:
  assumes LEX P f g
  obtains (eq) f = g | (neq) k where ∀ i<k. f i = g i and P (f k) (g k)
using assms by (auto simp: LEX-def)

lemma LEX-imp-less:
  assumes ∀ x∈A. ¬ P x x and f ∈ SEQ A ∨ g ∈ SEQ A
    and LEX P f g and ∀ i<k. f i = g i and f k ≠ g k
  shows P (f k) (g k)
using assms by (auto elim!: LEX-cases) (metis linorder-neqE-nat)+

end
```

2 Minimal elements of sets w.r.t. a well-founded and transitive relation

```
theory Minimal-Elements
imports
  Infinite-Sequences
  Open-Induction.Restricted-Predicates
begin

locale minimal-element =
  fixes P A
  assumes po: po-on P A
    and wf: wfp-on P A
begin

definition min-elt B = (SOME x. x ∈ B ∧ (∀ y ∈ A. P y x ⟶ y ∉ B))

lemma minimal:
  assumes x ∈ A and Q x
  shows ∃ y ∈ A. P⁼⁼ y x ∧ Q y ∧ (∀ z ∈ A. P z y ⟶ ¬ Q z)
using wf and assms
proof (induction rule: wfp-on-induct)
  case (less x)
```

then show *?case*
proof (*cases* $\forall y \in A.\ P\ y\ x \longrightarrow \neg\ Q\ y$)
case *True*
with *less* **show** *?thesis* **by** *blast*
next
case *False*
then obtain y **where** $y \in A$ **and** $P\ y\ x$ **and** $Q\ y$ **by** *blast*
with *less* **show** *?thesis*
using *po* [*THEN po-on-imp-transp-on*, *unfolded transp-on-def*, *rule-format*, *of* - $y\ x$] **by** *blast*
qed
qed

lemma *min-elt-ex*:
assumes $B \subseteq A$ **and** $B \neq \{\}$
shows $\exists x.\ x \in B \wedge (\forall y \in A.\ P\ y\ x \longrightarrow y \notin B)$
using *assms* **using** *minimal* [*of* - $\lambda x.\ x \in B$] **by** *auto*

lemma *min-elt-mem*:
assumes $B \subseteq A$ **and** $B \neq \{\}$
shows *min-elt* $B \in B$
using *someI-ex* [*OF min-elt-ex* [*OF assms*]] **by** (*auto simp*: *min-elt-def*)

lemma *min-elt-minimal*:
assumes $*$: $B \subseteq A$ $B \neq \{\}$
assumes $y \in A$ **and** $P\ y$ (*min-elt* B)
shows $y \notin B$
using *someI-ex* [*OF min-elt-ex* [*OF* $*$]] **and** *assms* **by** (*auto simp*: *min-elt-def*)

A lexicographically minimal sequence w.r.t. a given set of sequences C

fun *lexmin*
where
lexmin: *lexmin* C i = *min-elt* (*ith* (*eq-upto* C (*lexmin* C) i) i)
declare *lexmin* [*simp del*]

lemma *eq-upto-lexmin-non-empty*:
assumes $C \subseteq$ *SEQ* A **and** $C \neq \{\}$
shows *eq-upto* C (*lexmin* C) $i \neq \{\}$
proof (*induct* i)
case *0*
show *?case* **using** *assms* **by** *auto*
next
let *?A* = $\lambda i.$ *ith* (*eq-upto* C (*lexmin* C) i) i
case (*Suc* i)
then have *?A* $i \neq \{\}$ **by** *force*
moreover have *eq-upto* C (*lexmin* C) $i \subseteq$ *eq-upto* C (*lexmin* C) *0* **by** *auto*
ultimately have *?A* $i \subseteq A$ **and** *?A* $i \neq \{\}$ **using** *assms* **by** (*auto simp*: *ith-def*)
from *min-elt-mem* [*OF this*, *folded lexmin*]
obtain f **where** $f \in$ *eq-upto* C (*lexmin* C) (*Suc* i) **by** (*auto dest*: *eq-upto-Suc*)

then show *?case* **by** *blast*
qed

lemma *lexmin-SEQ-mem*:
assumes $C \subseteq SEQ\ A$ **and** $C \neq \{\}$
shows *lexmin* $C \in SEQ\ A$
proof –
{ fix i
let $?X = ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)\ i$
have $?X \subseteq A$ **using** *assms* **by** (*auto simp*: *ith-def*)
moreover have $?X \neq \{\}$ **using** *eq-upto-lexmin-non-empty* [*OF assms*] **by** *auto*
ultimately have *lexmin* $C\ i \in A$ **using** *min-elt-mem* [*of ?X*] **by** (*subst lexmin*) *blast* **}**
then show *?thesis* **by** *auto*
qed

lemma *non-empty-ith*:
assumes $C \subseteq SEQ\ A$ **and** $C \neq \{\}$
shows $ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)\ i \subseteq A$
and $ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)\ i \neq \{\}$
using *eq-upto-lexmin-non-empty* [*OF assms*, *of i*] **and** *assms* **by** (*auto simp*: *ith-def*)

lemma *lexmin-minimal*:
$C \subseteq SEQ\ A \Longrightarrow C \neq \{\} \Longrightarrow y \in A \Longrightarrow P\ y\ (lexmin\ C\ i) \Longrightarrow y \notin ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)\ i$
using *min-elt-minimal* [*OF non-empty-ith*, *folded lexmin*] **.**

lemma *lexmin-mem*:
$C \subseteq SEQ\ A \Longrightarrow C \neq \{\} \Longrightarrow lexmin\ C\ i \in ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)\ i$
using *min-elt-mem* [*OF non-empty-ith*, *folded lexmin*] **.**

lemma *LEX-chain-on-eq-upto-imp-ith-chain-on*:
assumes *chain-on* (*LEX P*) (*eq-upto C f i*) (*SEQ A*)
shows *chain-on P* (*ith* (*eq-upto C f i*) *i*) *A*
using *assms*
proof –
{ fix $x\ y$ **assume** $x \in ith\ (eq\text{-}upto\ C\ f\ i)\ i$ **and** $y \in ith\ (eq\text{-}upto\ C\ f\ i)\ i$
and $\neg\ P\ x\ y$ **and** $y \neq x$
then obtain $g\ h$ **where** $*$: $g \in eq\text{-}upto\ C\ f\ i\ h \in eq\text{-}upto\ C\ f\ i$
and [*simp*]: $x = g\ i\ y = h\ i$ **and** *eq*: $\forall j{<}i.\ g\ j = f\ j \wedge h\ j = f\ j$
by (*auto simp*: *ith-def eq-upto-def*)
with *assms* **and** ‹$y \neq x$› **consider** $LEX\ P\ g\ h \mid LEX\ P\ h\ g$ **by** (*force simp*: *chain-on-def*)
then have $P\ y\ x$
proof (*cases*)
assume $LEX\ P\ g\ h$
with *eq* **and** ‹$y \neq x$› **have** $P\ x\ y$ **using** *assms* **and** $*$
by (*auto simp*: *LEX-def*)
(*metis SEQ-iff chain-on-imp-subset linorder-neqE-nat minimal subsetCE*)

```
      with ‹¬ P x y› show P y x ..
    next
      assume LEX P h g
      with eq and ‹y ≠ x› show P y x using assms and ∗
        by (auto simp: LEX-def)
          (metis SEQ-iff chain-on-imp-subset linorder-neqE-nat minimal subsetCE)
    qed }
  then show ?thesis using assms by (auto simp: chain-on-def) blast
qed

end

end
```

3 Enumerations of Well-Ordered Sets in Increasing Order

```
theory Least-Enum
imports Main
begin

locale infinitely-many1 =
  fixes P :: 'a :: wellorder ⇒ bool
  assumes infm: ∀ i. ∃ j>i. P j
begin
```

Enumerate the elements of a well-ordered infinite set in increasing order.

```
fun enum :: nat ⇒ 'a where
  enum 0 = (LEAST n. P n) |
  enum (Suc i) = (LEAST n. n > enum i ∧ P n)

lemma enum-mono:
  shows enum i < enum (Suc i)
  using infm by (cases i, auto) (metis (lifting) LeastI)+

lemma enum-less:
  i < j ⟹ enum i < enum j
  using enum-mono by (metis lift-Suc-mono-less)

lemma enum-P:
  shows P (enum i)
  using infm by (cases i, auto) (metis (lifting) LeastI)+

end

locale infinitely-many2 =
  fixes P :: 'a :: wellorder ⇒ 'a ⇒ bool
    and N :: 'a
```

assumes *infm*: $\forall\, i \geq N.\ \exists\, j > i.\ P\ i\ j$
begin

Enumerate the elements of a well-ordered infinite set that form a chain w.r.t. a given predicate P starting from a given index N in increasing order.

fun *enumchain* :: *nat* ⇒ *'a* **where**
enumchain 0 = *N* |
enumchain (*Suc n*) = (*LEAST m*. *m* > *enumchain n* ∧ *P* (*enumchain n*) *m*)

lemma *enumchain-mono*:
shows *N* ≤ *enumchain i* ∧ *enumchain i* < *enumchain* (*Suc i*)
proof (*induct i*)
case *0*
have *enumchain 0* ≥ *N* **by** *simp*
moreover then have ∃ *m*>*enumchain 0*. *P* (*enumchain 0*) *m* **using** *infm* **by** *blast*
ultimately show *?case* **by** *auto* (*metis* (*lifting*) *LeastI*)
next
case (*Suc i*)
then have *N* ≤ *enumchain* (*Suc i*) **by** *auto*
moreover then have ∃ *m*>*enumchain* (*Suc i*). *P* (*enumchain* (*Suc i*)) *m* **using** *infm* **by** *blast*
ultimately show *?case* **by** (*auto*) (*metis* (*lifting*) *LeastI*)
qed

lemma *enumchain-chain*:
shows *P* (*enumchain i*) (*enumchain* (*Suc i*))
proof (*cases i*)
case *0*
moreover have ∃ *m*>*enumchain 0*. *P* (*enumchain 0*) *m* **using** *infm* **by** *auto*
ultimately show *?thesis* **by** *auto* (*metis* (*lifting*) *LeastI*)
next
case (*Suc i*)
moreover have *enumchain* (*Suc i*) > *N* **using** *enumchain-mono* **by** (*metis le-less-trans*)
moreover then have ∃ *m*>*enumchain* (*Suc i*). *P* (*enumchain* (*Suc i*)) *m* **using** *infm* **by** *auto*
ultimately show *?thesis* **by** (*auto*) (*metis* (*lifting*) *LeastI*)
qed

end

end

4 The Almost-Full Property

theory *Almost-Full*
imports
HOL−Library.Sublist

HOL−Library.Ramsey
Regular−Sets.Regexp-Method
Abstract−Rewriting.Seq
Least-Enum
Infinite-Sequences
Open-Induction.Restricted-Predicates
begin

lemma *le-Suc-eq′*:
$x \leq Suc\ y \longleftrightarrow x = 0 \vee (\exists x'.\ x = Suc\ x' \wedge x' \leq y)$
by (*cases x*) *auto*

lemma *ex-leq-Suc*:
$(\exists i{\leq}Suc\ j.\ P\ i) \longleftrightarrow P\ 0 \vee (\exists i{\leq}j.\ P\ (Suc\ i))$
by (*auto simp*: *le-Suc-eq′*)

lemma *ex-less-Suc*:
$(\exists i{<}Suc\ j.\ P\ i) \longleftrightarrow P\ 0 \vee (\exists i{<}j.\ P\ (Suc\ i))$
by (*auto simp*: *less-Suc-eq-0-disj*)

4.1 Basic Definitions and Facts

An infinite sequence is *good* whenever there are indices $i < j$ such that $P\ (f\ i)\ (f\ j)$.

definition *good* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$
where
$good\ P\ f \longleftrightarrow (\exists i\ j.\ i < j \wedge P\ (f\ i)\ (f\ j))$

A sequence that is not good is called *bad*.

abbreviation $bad\ P\ f \equiv \neg\ good\ P\ f$

lemma *goodI*:
$[\![i < j;\ P\ (f\ i)\ (f\ j)]\!] \Longrightarrow good\ P\ f$
by (*auto simp*: *good-def*)

lemma *goodE* [*elim*]:
$good\ P\ f \Longrightarrow (\bigwedge i\ j.\ [\![i < j;\ P\ (f\ i)\ (f\ j)]\!] \Longrightarrow Q) \Longrightarrow Q$
by (*auto simp*: *good-def*)

lemma *badE* [*elim*]:
$bad\ P\ f \Longrightarrow ((\bigwedge i\ j.\ i < j \Longrightarrow \neg\ P\ (f\ i)\ (f\ j)) \Longrightarrow Q) \Longrightarrow Q$
by (*auto simp*: *good-def*)

definition *almost-full-on* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
where
$almost\text{-}full\text{-}on\ P\ A \longleftrightarrow (\forall f \in SEQ\ A.\ good\ P\ f)$

lemma *almost-full-onI* [*Pure.intro*]:
  $(\bigwedge f.\ \forall\, i.\ f\ i \in A \Longrightarrow good\ P\ f) \Longrightarrow almost\text{-}full\text{-}on\ P\ A$
  unfolding *almost-full-on-def* **by** *blast*

lemma *almost-full-onD*:
  fixes $f :: nat \Rightarrow 'a$ **and** $A :: 'a\ set$
  assumes $almost\text{-}full\text{-}on\ P\ A$ **and** $\bigwedge i.\ f\ i \in A$
  obtains $i\ j$ **where** $i < j$ **and** $P\ (f\ i)\ (f\ j)$
  using *assms* **unfolding** *almost-full-on-def* **by** *blast*

4.2 An equivalent inductive definition

inductive *af* **for** *A*
  where
    now: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow P\ x\ y) \Longrightarrow af\ A\ P$
  | *later*: $(\bigwedge x.\ x \in A \Longrightarrow af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y)) \Longrightarrow af\ A\ P$

lemma *af-imp-almost-full-on*:
  assumes $af\ A\ P$
  shows $almost\text{-}full\text{-}on\ P\ A$
proof
  fix $f :: nat \Rightarrow 'a$ **assume** $\forall\, i.\ f\ i \in A$
  with *assms* **obtain** i **and** j **where** $i < j$ **and** $P\ (f\ i)\ (f\ j)$
  proof (*induct arbitrary*: *f thesis*)
    case (*later P*)
    define g **where** [*simp*]: $g\ i = f\ (Suc\ i)$ **for** i
    have $f\ 0 \in A$ **and** $\forall\, i.\ g\ i \in A$ **using** *later* **by** *auto*
    then obtain i **and** j **where** $i < j$ **and** $P\ (g\ i)\ (g\ j) \vee P\ (f\ 0)\ (g\ i)$ **using**
later **by** *blast*
    then consider $P\ (g\ i)\ (g\ j) \mid P\ (f\ 0)\ (g\ i)$ **by** *blast*
    then show *?case* **using** ‹$i < j$› **by** (*cases*) (*auto intro*: *later*)
  qed *blast*
  then show $good\ P\ f$ **by** (*auto simp*: *good-def*)
qed

lemma *af-mono*:
  assumes $af\ A\ P$
    and $\forall\, x\ y.\ x \in A \wedge y \in A \wedge P\ x\ y \longrightarrow Q\ x\ y$
  shows $af\ A\ Q$
  using *assms*
proof (*induct arbitrary*: *Q*)
  case (*now P*)
  then have $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ x\ y$ **by** *blast*
  then show *?case* **by** (*rule af.now*)
next
  case (*later P*)
  show *?case*
  proof (*intro af.later* [*of A Q*])

fix x **assume** $x \in A$
then show *af* A $(\lambda y\ z.\ Q\ y\ z \vee Q\ x\ y)$
using *later*(*3*) **by** (*intro later*(*2*) [*of x*]) *auto*
qed
qed

lemma *accessible-on-imp-af*:
assumes *accessible-on* $P\ A\ x$
shows *af* A $(\lambda u\ v.\ \neg\ P\ v\ u \vee \neg\ P\ u\ x)$
using *assms*
proof (*induct*)
case (*1 x*)
then have *af* A $(\lambda u\ v.\ (\neg\ P\ v\ u \vee \neg\ P\ u\ x) \vee \neg\ P\ u\ y \vee \neg\ P\ y\ x)$ **if** $y \in A$ **for** y
using *that* **by** (*cases* $P\ y\ x$) (*auto intro*: *af.now af-mono*)
then show *?case* **by** (*rule af.later*)
qed

lemma *wfp-on-imp-af*:
assumes *wfp-on* $P\ A$
shows *af* A $(\lambda x\ y.\ \neg\ P\ y\ x)$
using *assms* **by** (*auto simp*: *wfp-on-accessible-on-iff intro*: *accessible-on-imp-af af.later*)

lemma *af-leq*:
af UNIV $((\leq) :: nat \Rightarrow nat \Rightarrow bool)$
using *wf-less* [*folded wfp-def wfp-on-UNIV*, *THEN wfp-on-imp-af*] **by** (*simp add*: *not-less*)

definition *NOTAF* $A\ P = (SOME\ x.\ x \in A \wedge \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y))$

lemma *not-af*:
$\neg\ af\ A\ P \Longrightarrow (\exists\, x\ y.\ x \in A \wedge y \in A \wedge \neg\ P\ x\ y) \wedge (\exists\, x{\in}A.\ \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y))$
unfolding *af.simps* [*of A P*] **by** *blast*

fun F
where
$F\ A\ P\ 0 = NOTAF\ A\ P$
$|\ F\ A\ P\ (Suc\ i) = (let\ x = NOTAF\ A\ P\ in\ F\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y)\ i)$

lemma *almost-full-on-imp-af*:
assumes *af*: *almost-full-on* $P\ A$
shows *af* $A\ P$
proof (*rule ccontr*)
assume $\neg\ af\ A\ P$
then have $*$: $F\ A\ P\ n \in A \wedge$
$\neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee (\exists\, i{\le}n.\ P\ (F\ A\ P\ i)\ y) \vee (\exists\, j{\le}n.\ \exists\, i.\ i < j \wedge P\ (F\ A\ P\ i)\ (F\ A\ P\ j)))$ **for** n
proof (*induct n arbitrary*: P)

case *0*
from ‹¬ *af A P*› **have** $\exists x.\ x \in A \wedge \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y)$ **by** (*auto intro*: *af.intros*)
then have $NOTAF\ A\ P \in A \wedge \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ (NOTAF\ A\ P)\ y)$ **unfolding** *NOTAF-def* **by** (*rule someI-ex*)
with *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
from ‹¬ *af A P*› **have** $\exists x.\ x \in A \wedge \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ x\ y)$ **by** (*auto intro*: *af.intros*)
then have $NOTAF\ A\ P \in A \wedge \neg\ af\ A\ (\lambda y\ z.\ P\ y\ z \vee P\ (NOTAF\ A\ P)\ y)$ **unfolding** *NOTAF-def* **by** (*rule someI-ex*)
from *Suc*(*1*) [*OF this* [*THEN conjunct2*]]
show *?case*
by (*fastforce simp*: *ex-leq-Suc ex-less-Suc elim*!: *back-subst* [**where** $P = \lambda x.\ \neg\ af\ A\ x$])
qed
then have $F\ A\ P \in SEQ\ A$ **by** *auto*
from *af* [*unfolded almost-full-on-def*, *THEN bspec*, *OF this*] **and** *not-af* [*OF* ∗ [*THEN conjunct2*]]
show *False* **unfolding** *good-def* **by** *blast*
qed

hide-const *NOTAF F*

lemma *almost-full-on-UNIV*:
almost-full-on (λ- -. *True*) *UNIV*
by (*auto simp*: *almost-full-on-def good-def*)

lemma *almost-full-on-imp-reflp-on*:
assumes *almost-full-on P A*
shows *reflp-on A P*
using *assms* **by** (*auto simp*: *almost-full-on-def reflp-on-def*)

lemma *almost-full-on-subset*:
$A \subseteq B \Longrightarrow$ *almost-full-on P B* $\Longrightarrow$ *almost-full-on P A*
by (*auto simp*: *almost-full-on-def*)

lemma *almost-full-on-mono*:
assumes $A \subseteq B$ **and** $\bigwedge x\ y.\ Q\ x\ y \Longrightarrow P\ x\ y$
and *almost-full-on Q B*
shows *almost-full-on P A*
using *assms* **by** (*metis almost-full-on-def almost-full-on-subset good-def*)

Every sequence over elements of an almost-full set has a homogeneous subsequence.

lemma *almost-full-on-imp-homogeneous-subseq*:
assumes *almost-full-on P A*
and $\forall i{::}nat.\ f\ i \in A$

shows $\exists \varphi$::*nat* $\Rightarrow$ *nat*. $\forall\, i\, j.\ i < j \longrightarrow \varphi\ i < \varphi\ j \wedge P\ (f\ (\varphi\ i))\ (f\ (\varphi\ j))$
proof −
  define X **where** $X = \{\{i, j\} \mid i\ j$::*nat*. $i < j \wedge P\ (f\ i)\ (f\ j)\}$
  define Y **where** $Y = -\ X$
  define h **where** $h = (\lambda Z.\ \mathit{if}\ Z \in X\ \mathit{then}\ 0\ \mathit{else}\ \mathit{Suc}\ 0)$

  have [*iff*]: $\bigwedge x\ y.\ h\ \{x, y\} = 0 \longleftrightarrow \{x, y\} \in X$ **by** (*auto simp*: *h-def*)
  have [*iff*]: $\bigwedge x\ y.\ h\ \{x, y\} = \mathit{Suc}\ 0 \longleftrightarrow \{x, y\} \in Y$ **by** (*auto simp*: *h-def Y-def*)

  have $\forall x{\in}\mathit{UNIV}.\ \forall y{\in}\mathit{UNIV}.\ x \neq y \longrightarrow h\ \{x, y\} < 2$ **by** (*simp add*: *h-def*)
  from *Ramsey2* [*OF infinite-UNIV-nat this*] **obtain** $I\ c$
    where *infinite* I **and** $c < 2$
    and $*$: $\forall x{\in}I.\ \forall y{\in}I.\ x \neq y \longrightarrow h\ \{x, y\} = c$ **by** *blast*
  then interpret *infinitely-many1* $\lambda i.\ i \in I$
    by (*unfold-locales*) (*simp add*: *infinite-nat-iff-unbounded*)

  have $c = 0 \vee c = 1$ **using** ‹$c < 2$› **by** *arith*
  then show *?thesis*
  proof
    assume [*simp*]: $c = 0$
    have $\forall\, i\, j.\ i < j \longrightarrow P\ (f\ (\mathit{enum}\ i))\ (f\ (\mathit{enum}\ j))$
    proof (*intro allI impI*)
      fix $i\ j$ :: *nat*
      assume $i < j$
      from $*$ **and** *enum-P* **and** *enum-less* [*OF* ‹$i < j$›] **have** $\{\mathit{enum}\ i, \mathit{enum}\ j\} \in$
X **by** *auto*
      with *enum-less* [*OF* ‹$i < j$›]
        show $P\ (f\ (\mathit{enum}\ i))\ (f\ (\mathit{enum}\ j))$ **by** (*auto simp*: *X-def doubleton-eq-iff*)
    qed
    then show *?thesis* **using** *enum-less* **by** *blast*
  next
    assume [*simp*]: $c = 1$
    have $\forall\, i\, j.\ i < j \longrightarrow \neg\ P\ (f\ (\mathit{enum}\ i))\ (f\ (\mathit{enum}\ j))$
    proof (*intro allI impI*)
      fix $i\ j$ :: *nat*
      assume $i < j$
      from $*$ **and** *enum-P* **and** *enum-less* [*OF* ‹$i < j$›] **have** $\{\mathit{enum}\ i, \mathit{enum}\ j\} \in$
Y **by** *auto*
      with *enum-less* [*OF* ‹$i < j$›]
        show $\neg\ P\ (f\ (\mathit{enum}\ i))\ (f\ (\mathit{enum}\ j))$ **by** (*auto simp*: *Y-def X-def double-
ton-eq-iff*)
    qed
    then have $\neg$ *good* $P\ (f \circ \mathit{enum})$ **by** *auto*
    moreover have $\forall\, i.\ f\ (\mathit{enum}\ i) \in A$ **using** *assms* **by** *auto*
    ultimately show *?thesis* **using** ‹*almost-full-on* $P\ A$› **by** (*simp add*: *almost-full-on-def*)
  qed
qed

Almost full relations do not admit infinite antichains.

lemma *almost-full-on-imp-no-antichain-on*:
  assumes *almost-full-on P A*
  shows ¬ *antichain-on P f A*
proof
  assume ∗: *antichain-on P f A*
  then have $\forall\, i.\ f\ i \in A$ **by** *simp*
  with *assms* **have** *good P f* **by** (*auto simp*: *almost-full-on-def*)
  then obtain *i j* **where** $i < j$ **and** *P* (*f i*) (*f j*)
    unfolding *good-def* **by** *auto*
  moreover with ∗ **have** *incomparable P* (*f i*) (*f j*) **by** *auto*
  ultimately show *False* **by** *blast*
qed

If the image of a function is almost-full then also its preimage is almost-full.

lemma *almost-full-on-map*:
  assumes *almost-full-on Q B*
    and *h* ‘ $A \subseteq B$
  shows *almost-full-on* ($\lambda x\ y.\ Q\ (h\ x)\ (h\ y)$) *A* (**is** *almost-full-on ?P A*)
proof
  fix *f*
  assume $\forall\, i{::}nat.\ f\ i \in A$
  then have $\bigwedge i.\ h\ (f\ i) \in B$ **using** ‹*h* ‘ $A \subseteq B$› **by** *auto*
  with ‹*almost-full-on Q B*› [*unfolded almost-full-on-def*, *THEN bspec*, *of h* ∘ *f*]
    show *good ?P f* **unfolding** *good-def comp-def* **by** *blast*
qed

The homomorphic image of an almost-full set is almost-full.

lemma *almost-full-on-hom*:
  fixes $h :: {'a} \Rightarrow {'b}$
  assumes *hom*: $\bigwedge x\ y.\ [\![x \in A;\ y \in A;\ P\ x\ y]\!] \Longrightarrow Q\ (h\ x)\ (h\ y)$
    and *af*: *almost-full-on P A*
  shows *almost-full-on Q* (*h* ‘ *A*)
proof
  fix $f :: nat \Rightarrow {'b}$
  assume $\forall\, i.\ f\ i \in h$ ‘ *A*
  then have $\forall\, i.\ \exists x.\ x \in A \wedge f\ i = h\ x$ **by** (*auto simp*: *image-def*)
  from *choice* [*OF this*] **obtain** *g*
    where ∗: $\forall\, i.\ g\ i \in A \wedge f\ i = h\ (g\ i)$ **by** *blast*
  show *good Q f*
  proof (*rule ccontr*)
    assume *bad*: *bad Q f*
    { fix *i j* :: *nat*
      assume $i < j$
      from *bad* **have** ¬ *Q* (*f i*) (*f j*) **using** ‹$i < j$› **by** (*auto simp*: *good-def*)
      with *hom* **have** ¬ *P* (*g i*) (*g j*) **using** ∗ **by** *auto* **}**
    then have *bad P g* **by** (*auto simp*: *good-def*)
    with *af* **and** ∗ **show** *False* **by** (*auto simp*: *good-def almost-full-on-def*)
  qed
qed

The monomorphic preimage of an almost-full set is almost-full.

lemma *almost-full-on-mon*:
  assumes *mon*: $\bigwedge x\ y.\ [\![x \in A;\ y \in A]\!] \Longrightarrow P\ x\ y = Q\ (h\ x)\ (h\ y)$ *bij-betw* $h\ A\ B$
    and *af*: *almost-full-on* $Q\ B$
  shows *almost-full-on* $P\ A$
proof
  fix $f :: nat \Rightarrow {'a}$
  assume $*$: $\forall\, i.\ f\ i \in A$
  then have $**$: $\forall\, i.\ (h \circ f)\ i \in B$ **using** *mon* **by** (*auto simp*: *bij-betw-def*)
  show *good* $P\ f$
  proof (*rule ccontr*)
    assume *bad*: *bad* $P\ f$
    { fix $i\ j :: nat$
      assume $i < j$
      from *bad* **have** $\neg\ P\ (f\ i)\ (f\ j)$ **using** $\langle i < j\rangle$ **by** (*auto simp*: *good-def*)
      with *mon* **have** $\neg\ Q\ (h\ (f\ i))\ (h\ (f\ j))$
        using $*$ **by** (*auto simp*: *bij-betw-def inj-on-def*) **}**
    then have *bad* $Q\ (h \circ f)$ **by** (*auto simp*: *good-def*)
    with *af* **and** $**$ **show** *False* **by** (*auto simp*: *good-def almost-full-on-def*)
  qed
qed

Every total and well-founded relation is almost-full.

lemma *total-on-and-wfp-on-imp-almost-full-on*:
  assumes *totalp-on* $A\ P$ **and** *wfp-on* $P\ A$
  shows *almost-full-on* $P^{==}\ A$
proof (*rule ccontr*)
  assume $\neg$ *almost-full-on* $P^{==}\ A$
  then obtain $f :: nat \Rightarrow {'a}$ **where** $*$: $\bigwedge i.\ f\ i \in A$
    and $\forall\, i\ j.\ i < j \longrightarrow \neg\ P^{==}\ (f\ i)\ (f\ j)$
    unfolding *almost-full-on-def* **by** (*auto dest*: *badE*)
  with $\langle$*totalp-on* $A\ P\rangle$ **have** $\forall\, i\ j.\ i < j \longrightarrow P\ (f\ j)\ (f\ i)$
    unfolding *totalp-on-def* **by** *blast*
  then have $\bigwedge i.\ P\ (f\ (Suc\ i))\ (f\ i)$ **by** *auto*
  with $\langle$*wfp-on* $P\ A\rangle$ **and** $*$ **show** *False*
    unfolding *wfp-on-def* **by** *blast*
qed

lemma *Nil-imp-good-list-emb* [*simp*]:
  assumes $f\ i = []$
  shows *good* (*list-emb* P) f
proof (*rule ccontr*)
  assume *bad* (*list-emb* P) f
  moreover have (*list-emb* P) $(f\ i)\ (f\ (Suc\ i))$
    unfolding *assms* **by** *auto*
  ultimately show *False*
    unfolding *good-def* **by** *auto*
qed

lemma *ne-lists*:
  assumes $xs \neq []$ **and** $xs \in$ *lists* A
  shows *hd* $xs \in A$ **and** *tl* $xs \in$ *lists* A
  using *assms* **by** (*case-tac* [!] *xs*) *simp-all*

lemma *list-emb-eq-length-induct* [*consumes 2*, *case-names Nil Cons*]:
  assumes *length xs* = *length ys*
    and *list-emb P xs ys*
    and *Q* [] []
    and $\bigwedge x\ y\ xs\ ys.\ [\![P\ x\ y;\ \textit{list-emb}\ P\ xs\ ys;\ Q\ xs\ ys]\!] \Longrightarrow Q\ (x\#xs)\ (y\#ys)$
  shows *Q xs ys*
  using *assms*(*2*, *1*, *3*−) **by** (*induct*) (*auto dest*: *list-emb-length*)

lemma *list-emb-eq-length-P*:
  assumes *length xs* = *length ys*
    and *list-emb P xs ys*
  shows $\forall\, i<\textit{length}\ xs.\ P\ (xs\ !\ i)\ (ys\ !\ i)$
using *assms*
proof (*induct rule*: *list-emb-eq-length-induct*)
  case (*Cons x y xs ys*)
  show *?case*
  proof (*intro allI impI*)
    fix *i* **assume** $i < \textit{length}\ (x\ \#\ xs)$
    with *Cons* **show** $P\ ((x\#xs)!i)\ ((y\#ys)!i)$
      by (*cases i*) *simp-all*
  qed
qed *simp*

4.3 Special Case: Finite Sets

Every reflexive relation on a finite set is almost-full.

lemma *finite-almost-full-on*:
  assumes *finite*: *finite A*
    and *refl*: *reflp-on A P*
  shows *almost-full-on P A*
proof
  fix $f :: \textit{nat} \Rightarrow 'a$
  assume $*$: $\forall\, i.\ f\ i \in A$
  let *?I* = *UNIV*::*nat set*
  have f ‘ *?I* $\subseteq A$ **using** $*$ **by** *auto*
  with *finite* **and** *finite-subset* **have** *1*: *finite* (f ‘ *?I*) **by** *blast*
  have *infinite ?I* **by** *auto*
  from *pigeonhole-infinite* [*OF this 1*]
    obtain *k* **where** *infinite* $\{j.\ f\ j = f\ k\}$ **by** *auto*
  then obtain *l* **where** $k < l$ **and** $f\ l = f\ k$
    unfolding *infinite-nat-iff-unbounded* **by** *auto*
  then have $P\ (f\ k)\ (f\ l)$ **using** *refl* **and** $*$ **by** (*auto simp*: *reflp-on-def*)
  with ‹$k < l$› **show** *good P f* **by** (*auto simp*: *good-def*)
qed

lemma *eq-almost-full-on-finite-set*:
  assumes *finite A*
  shows *almost-full-on* $(=)$ *A*
  using *finite-almost-full-on* [*OF assms*, *of* $(=)$]
  by (*auto simp*: *reflp-on-def*)

4.4 Further Results

lemma *af-trans-extension-imp-wf*:
  assumes *subrel*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow Q\ x\ y$
    and *af*: *almost-full-on P A*
    and *trans*: *transp-on A Q*
  shows *wfp-on* (*strict Q*) *A*
proof (*unfold wfp-on-def*, *rule notI*)
  assume $\exists f.\ \forall i.\ f\ i \in A \wedge$ *strict* $Q\ (f\ (Suc\ i))\ (f\ i)$
  then obtain *f* **where** $*$: $\forall i.\ f\ i \in A \wedge ((strict\ Q)^{-1-1})\ (f\ i)\ (f\ (Suc\ i))$ **by** *blast*
  from *chain-transp-on-less*[*OF this*]
   have $\forall i\ j.\ i < j \longrightarrow \neg\ Q\ (f\ i)\ (f\ j)$ **using** *trans* **using** *transp-on-conversep transp-on-strict* **by** *blast*
  with *subrel* **have** $\forall i\ j.\ i < j \longrightarrow \neg\ P\ (f\ i)\ (f\ j)$ **by** *blast*
  with *af* **show** *False*
    using $*$ **by** (*auto simp*: *almost-full-on-def good-def*)
qed

lemma *af-trans-imp-wf*:
  assumes *almost-full-on P A*
    and *transp-on A P*
  shows *wfp-on* (*strict P*) *A*
  using *assms* **by** (*intro af-trans-extension-imp-wf*)

lemma *wf-and-no-antichain-imp-qo-extension-wf*:
  assumes *wf*: *wfp-on* (*strict P*) *A*
    and *anti*: $\neg\ (\exists f.$ *antichain-on P f A*)
    and *subrel*: $\forall x{\in}A.\ \forall y{\in}A.\ P\ x\ y \longrightarrow Q\ x\ y$
    and *qo*: *qo-on Q A*
  shows *wfp-on* (*strict Q*) *A*
proof (*rule ccontr*)
  have *transp-on A* (*strict Q*)
    using *qo* **unfolding** *qo-on-def transp-on-def* **by** *blast*
  then have $*$: *transp-on A* $((strict\ Q)^{-1-1})$ **by** *simp*
  assume $\neg$ *wfp-on* (*strict Q*) *A*
  then obtain $f :: nat \Rightarrow 'a$ **where** *A*: $\bigwedge i.\ f\ i \in A$
    and $\forall i.$ *strict* $Q\ (f\ (Suc\ i))\ (f\ i)$ **unfolding** *wfp-on-def* **by** *blast+*
  then have $\forall i.\ f\ i \in A \wedge ((strict\ Q)^{-1-1})\ (f\ i)\ (f\ (Suc\ i))$ **by** *auto*
  from *chain-transp-on-less* [*OF this* $*$]
    have $*$: $\bigwedge i\ j.\ i < j \Longrightarrow \neg\ P\ (f\ i)\ (f\ j)$
    using *subrel* **and** *A* **by** *blast*
  show *False*

proof (*cases*)
assume $\exists k.\ \forall i>k.\ \exists j>i.\ P\ (f\ j)\ (f\ i)$
then obtain k **where** $\forall i>k.\ \exists j>i.\ P\ (f\ j)\ (f\ i)$ **by** *auto*
from *subchain* [*of k - f*, *OF this*] **obtain** g
where $\bigwedge i\ j.\ i < j \Longrightarrow g\ i < g\ j$
and $\bigwedge i.\ P\ (f\ (g\ (Suc\ i)))\ (f\ (g\ i))$ **by** *auto*
with $*$ **have** $\bigwedge i.\ strict\ P\ (f\ (g\ (Suc\ i)))\ (f\ (g\ i))$ **by** *blast*
with *wf* [*unfolded wfp-on-def not-ex*, *THEN spec*, *of* $\lambda i.\ f\ (g\ i)$] **and** A
show *False* **by** *fast*
next
assume $\neg\ (\exists k.\ \forall i>k.\ \exists j>i.\ P\ (f\ j)\ (f\ i))$
then have $\forall k.\ \exists i>k.\ \forall j>i.\ \neg\ P\ (f\ j)\ (f\ i)$ **by** *auto*
from *choice* [*OF this*] **obtain** h
where $\forall k.\ h\ k > k$
and $**$: $\forall k.\ (\forall j>h\ k.\ \neg\ P\ (f\ j)\ (f\ (h\ k)))$ **by** *auto*
define φ **where** [*simp*]: $\varphi = (\lambda i.\ (h\ \widehat{}\widehat{}\ Suc\ i)\ 0)$
have $\bigwedge i.\ \varphi\ i < \varphi\ (Suc\ i)$
using ‹$\forall k.\ h\ k > k$› **by** (*induct-tac i*) *auto*
then have *mono*: $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$ **by** (*metis lift-Suc-mono-less*)
then have $\forall i\ j.\ i < j \longrightarrow \neg\ P\ (f\ (\varphi\ j))\ (f\ (\varphi\ i))$
using $**$ **by** *auto*
with *mono* [*THEN* $*$]
have $\forall i\ j.\ i < j \longrightarrow incomparable\ P\ (f\ (\varphi\ j))\ (f\ (\varphi\ i))$ **by** *blast*
moreover have $\exists i\ j.\ i < j \wedge \neg\ incomparable\ P\ (f\ (\varphi\ i))\ (f\ (\varphi\ j))$
using *anti* [*unfolded not-ex*, *THEN spec*, *of* $\lambda i.\ f\ (\varphi\ i)$] **and** A **by** *blast*
ultimately show *False* **by** *blast*
qed
qed

lemma *every-qo-extension-wf-imp-af*:
assumes *ext*: $\forall Q.\ (\forall x{\in}A.\ \forall y{\in}A.\ P\ x\ y \longrightarrow Q\ x\ y)\ \wedge$
qo-on Q A $\longrightarrow$ *wfp-on* (*strict Q*) A
and *qo-on P A*
shows *almost-full-on P A*
proof
from ‹*qo-on P A*›
have *refl*: *reflp-on A P*
and *trans*: *transp-on A P*
by (*auto intro*: *qo-on-imp-reflp-on qo-on-imp-transp-on*)

fix $f :: nat \Rightarrow 'a$
assume $\forall i.\ f\ i \in A$
then have A: $\bigwedge i.\ f\ i \in A$ **..**
show *good P f*
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have *bad*: $\forall i\ j.\ i < j \longrightarrow \neg\ P\ (f\ i)\ (f\ j)$ **by** (*auto simp*: *good-def*)
then have $*$: $\bigwedge i\ j.\ P\ (f\ i)\ (f\ j) \Longrightarrow i \geq j$ **by** (*metis not-le-imp-less*)

define D **where** [*simp*]: $D = (\lambda x\ y.\ \exists\, i.\ x = f\ (Suc\ i) \wedge y = f\ i)$
define P' **where** $P' = restrict\text{-}to\ P\ A$
define Q **where** [*simp*]: $Q = (sup\ P'\ D)^{**}$

have $**$: $\bigwedge i\ j.\ (D\ OO\ P'^{**})^{++}\ (f\ i)\ (f\ j) \Longrightarrow i > j$
proof −
  fix $i\ j$
  assume $(D\ OO\ P'^{**})^{++}\ (f\ i)\ (f\ j)$
  then show $i > j$
    apply (*induct f i f j arbitrary*: *j*)
    apply (*insert A*, *auto dest*!: ∗ *simp*: *P′-def reflp-on-restrict-to-rtranclp* [*OF refl trans*])
    apply (*metis* ∗ *dual-order.strict-trans1 less-Suc-eq-le refl reflp-on-def*)
    by (*metis le-imp-less-Suc less-trans*)
qed

have $\forall\, x{\in}A.\ \forall\, y{\in}A.\ P\ x\ y \longrightarrow Q\ x\ y$ **by** (*auto simp*: *P′-def*)
moreover have *qo-on Q A* **by** (*auto simp*: *qo-on-def reflp-on-def transp-on-def*)
ultimately have *wfp-on* (*strict Q*) *A*
    using *ext* [*THEN spec*, *of Q*] **by** *blast*
moreover have $\forall\, i.\ f\ i \in A \wedge strict\ Q\ (f\ (Suc\ i))\ (f\ i)$
proof
  fix i
  have $\neg\ Q\ (f\ i)\ (f\ (Suc\ i))$
  proof
    assume $Q\ (f\ i)\ (f\ (Suc\ i))$
    then have $(sup\ P'\ D)^{**}\ (f\ i)\ (f\ (Suc\ i))$ **by** *auto*
    moreover have $(sup\ P'\ D)^{**} = sup\ (P'^{**})\ (P'^{**}\ OO\ (D\ OO\ P'^{**})^{++})$
    proof −
      have $\bigwedge A\ B.\ (A \cup B)^* = A^* \cup A^*\ O\ (B\ O\ A^*)^+$ **by** *regexp*
      from *this* [*to-pred*] **show** *?thesis* **by** *blast*
    qed
    ultimately have $sup\ (P'^{**})\ (P'^{**}\ OO\ (D\ OO\ P'^{**})^{++})\ (f\ i)\ (f\ (Suc\ i))$ **by** *simp*
    then have $(P'^{**}\ OO\ (D\ OO\ P'^{**})^{++})\ (f\ i)\ (f\ (Suc\ i))$ **by** *auto*
    then have $Suc\ i < i$
      using $**$ **apply** *auto*
    by (*metis* (*lifting*, *mono-tags*) *less-le relcompp.relcompI tranclp-into-tranclp2*)
    then show *False* **by** *auto*
  qed
  with *A* [*of i*] **show** $f\ i \in A \wedge strict\ Q\ (f\ (Suc\ i))\ (f\ i)$ **by** *auto*
qed
ultimately show *False* **unfolding** *wfp-on-def* **by** *blast*
qed
qed

end

5 Constructing Minimal Bad Sequences

theory *Minimal-Bad-Sequences*
imports
  Almost-Full
  Minimal-Elements
begin

A locale capturing the construction of minimal bad sequences over values from A. Where minimality is to be understood w.r.t. *size* of an element.

locale *mbs* =
  fixes $A :: ('a :: size)\ set$
begin

Since the *size* is a well-founded measure, whenever some element satisfies a property P, then there is a size-minimal such element.

lemma *minimal*:
  assumes $x \in A$ **and** $P\ x$
  shows $\exists\, y \in A.\ size\ y \leq size\ x \wedge P\ y \wedge (\forall\, z \in A.\ size\ z < size\ y \longrightarrow \neg\ P\ z)$
using *assms*
proof (*induction x taking*: *size rule*: *measure-induct*)
  case (*1 x*)
  then show *?case*
  proof (*cases* $\forall\, y \in A.\ size\ y < size\ x \longrightarrow \neg\ P\ y$)
    case *True*
    with *1* **show** *?thesis* **by** *blast*
  next
    case *False*
    then obtain y **where** $y \in A$ **and** $size\ y < size\ x$ **and** $P\ y$ **by** *blast*
    with *1.IH* **show** *?thesis* **by** (*fastforce elim*!: *order-trans*)
  qed
qed

lemma *less-not-eq* [*simp*]:
  $x \in A \Longrightarrow size\ x < size\ y \Longrightarrow x = y \Longrightarrow False$
  by *simp*

The set of all bad sequences over A.

definition $BAD\ P = \{f \in SEQ\ A.\ bad\ P\ f\}$

lemma *BAD-iff* [*iff*]:
  $f \in BAD\ P \longleftrightarrow (\forall\, i.\ f\ i \in A) \wedge bad\ P\ f$
  by (*auto simp*: *BAD-def*)

A partial order on infinite bad sequences.

definition $geseq :: ((nat \Rightarrow 'a) \times (nat \Rightarrow 'a))\ set$
where
  $geseq =$

$\{(f,\ g).\ f \in SEQ\ A \wedge g \in SEQ\ A \wedge (f = g \vee (\exists\, i.\ size\ (g\ i) < size\ (f\ i) \wedge (\forall\, j < i.\ f\, j = g\ j)))\}$

The strict part of the above order.

definition *gseq* :: $((nat \Rightarrow 'a) \times (nat \Rightarrow 'a))$ *set* **where**
$gseq = \{(f,\ g).\ f \in SEQ\ A \wedge g \in SEQ\ A \wedge (\exists\, i.\ size\ (g\ i) < size\ (f\ i) \wedge (\forall\, j < i.\ f\, j = g\ j))\}$

lemma *geseq-iff*:
$(f,\ g) \in geseq \longleftrightarrow$
$f \in SEQ\ A \wedge g \in SEQ\ A \wedge (f = g \vee (\exists\, i.\ size\ (g\ i) < size\ (f\ i) \wedge (\forall\, j < i.\ f\, j = g\ j)))$
by (*auto simp*: *geseq-def*)

lemma *gseq-iff*:
$(f,\ g) \in gseq \longleftrightarrow f \in SEQ\ A \wedge g \in SEQ\ A \wedge (\exists\, i.\ size\ (g\ i) < size\ (f\ i) \wedge (\forall\, j < i.\ f\, j = g\ j))$
by (*auto simp*: *gseq-def*)

lemma *geseqE*:
assumes $(f,\ g) \in geseq$
and $[\![\forall\, i.\ f\ i \in A;\ \forall\, i.\ g\ i \in A;\ f = g]\!] \Longrightarrow Q$
and $\bigwedge i.\ [\![\forall\, i.\ f\ i \in A;\ \forall\, i.\ g\ i \in A;\ size\ (g\ i) < size\ (f\ i);\ \forall\, j < i.\ f\, j = g\ j]\!] \Longrightarrow Q$
shows Q
using *assms* **by** (*auto simp*: *geseq-iff*)

lemma *gseqE*:
assumes $(f,\ g) \in gseq$
and $\bigwedge i.\ [\![\forall\, i.\ f\ i \in A;\ \forall\, i.\ g\ i \in A;\ size\ (g\ i) < size\ (f\ i);\ \forall\, j < i.\ f\, j = g\ j]\!] \Longrightarrow Q$
shows Q
using *assms* **by** (*auto simp*: *gseq-iff*)

sublocale *min-elt-size?*: *minimal-element measure-on size UNIV A*
rewrites *measure-on size UNIV* $\equiv \lambda x\ y.\ size\ x < size\ y$
apply (*unfold-locales*)
apply (*auto simp*: *po-on-def irreflp-on-def transp-on-def simp del*: *wfp-on-UNIV intro*: *wfp-on-subset*)
apply (*auto simp*: *measure-on-def inv-image-betw-def*)
done

context
fixes $P :: 'a \Rightarrow 'a \Rightarrow bool$
begin

A lower bound to all sequences in a set of sequences B.

abbreviation $lb \equiv lexmin\ (BAD\ P)$

lemma *eq-upto-BAD-mem*:
  assumes $f \in$ *eq-upto* (*BAD P*) *g i*
  shows *f j* $\in A$
  using *assms* **by** (*auto*)

Assume that there is some infinite bad sequence *h*.

context
  fixes *h* :: *nat* $\Rightarrow$ *'a*
  assumes *BAD-ex*: $h \in$ *BAD P*
begin

When there is a bad sequence, then filtering *BAD P* w.r.t. positions in *lb* never yields an empty set of sequences.

lemma *eq-upto-BAD-non-empty*:
  eq-upto (*BAD P*) *lb i* $\neq$ {}
using *eq-upto-lexmin-non-empty* [*of BAD P*] **and** *BAD-ex* **by** *auto*

lemma *non-empty-ith*:
  shows *ith* (*eq-upto* (*BAD P*) *lb i*) *i* $\subseteq A$
  and *ith* (*eq-upto* (*BAD P*) *lb i*) *i* $\neq$ {}
  using *eq-upto-BAD-non-empty* [*of i*] **by** *auto*

lemmas
  lb-minimal = *min-elt-minimal* [*OF non-empty-ith*, *folded lexmin*] **and**
  lb-mem = *min-elt-mem* [*OF non-empty-ith*, *folded lexmin*]

lb is a infinite bad sequence.

lemma *lb-BAD*:
  lb $\in$ *BAD P*
proof −
  have ∗: $\bigwedge$*j*. *lb j* $\in$ *ith* (*eq-upto* (*BAD P*) *lb j*) *j* **by** (*rule lb-mem*)
  then have $\forall$ *i*. *lb i* $\in A$ **by** (*auto simp*: *ith-conv*) (*metis eq-upto-BAD-mem*)
  moreover
  { assume *good P lb*
    then obtain *i j* **where** $i < j$ **and** *P* (*lb i*) (*lb j*) **by** (*auto simp*: *good-def*)
    from ∗ **have** *lb j* $\in$ *ith* (*eq-upto* (*BAD P*) *lb j*) *j* **by** (*auto*)
    then obtain *g* **where** $g \in$ *eq-upto* (*BAD P*) *lb j* **and** *g j* = *lb j* **by** *force*
    then have $\forall k \leq j$. *g k* = *lb k* **by** (*auto simp*: *order-le-less*)
    with ‹$i < j$› **and** ‹*P* (*lb i*) (*lb j*)› **have** *P* (*g i*) (*g j*) **by** *auto*
    with ‹$i < j$› **have** *good P g* **by** (*auto simp*: *good-def*)
    with ‹$g \in$ *eq-upto* (*BAD P*) *lb j*› **have** *False* **by** *auto* **}**
  ultimately show *?thesis* **by** *blast*
qed

There is no infinite bad sequence that is strictly smaller than *lb*.

lemma *lb-lower-bound*:
  $\forall$ *g*. (*lb*, *g*) $\in$ *gseq* $\longrightarrow$ *g* $\notin$ *BAD P*
proof (*intro allI impI*)

fix g
assume $(lb, g) \in gseq$
then obtain i **where** $g\ i \in A$ **and** $size\ (g\ i) < size\ (lb\ i)$
and $\forall j < i.\ lb\ j = g\ j$ **by** $(auto\ simp{:}\ gseq\text{-}iff)$
moreover with *lb-minimal*
have $g\ i \notin ith\ (eq\text{-}upto\ (BAD\ P)\ lb\ i)\ i$ **by** *auto*
ultimately show $g \notin BAD\ P$ **by** *blast*
qed

If there is at least one bad sequence, then there is also a minimal one.

lemma *lower-bound-ex*:
$\exists f \in BAD\ P.\ \forall g.\ (f, g) \in gseq \longrightarrow g \notin BAD\ P$
using *lb-BAD* **and** *lb-lower-bound* **by** *blast*

lemma *gseq-conv*:
$(f, g) \in gseq \longleftrightarrow f \neq g \wedge (f, g) \in geseq$
by $(auto\ simp{:}\ gseq\text{-}def\ geseq\text{-}def\ dest{:}\ less\text{-}not\text{-}eq)$

There is a minimal bad sequence.

lemma *mbs*:
$\exists f \in BAD\ P.\ \forall g.\ (f, g) \in gseq \longrightarrow good\ P\ g$
using *lower-bound-ex* **by** $(auto\ simp{:}\ gseq\text{-}conv\ geseq\text{-}iff)$

end

end

end

end

6 A Proof of Higman's Lemma via Open Induction

theory *Higman-OI*
imports
Open-Induction.Open-Induction
Minimal-Elements
Almost-Full
begin

6.1 Some facts about the suffix relation

lemma *wfp-on-strict-suffix*:
wfp-on strict-suffix A
by (*rule wfp-on-mono* [*OF subset-refl*, *of - - measure-on length A*])
(*auto simp*: *strict-suffix-def suffix-def*)

lemma *po-on-strict-suffix*:

po-on strict-suffix A
by (*force simp*: *strict-suffix-def po-on-def transp-on-def irreflp-on-def*)

6.2 Lexicographic Order on Infinite Sequences

lemma *antisymp-on-LEX*:
  assumes *irreflp-on A P* **and** *antisymp-on A P*
  shows *antisymp-on* (*SEQ A*) (*LEX P*)
proof (*rule antisymp-onI*)
  fix *f g* **assume** *SEQ*: $f \in SEQ\ A$ $g \in SEQ\ A$ **and** *LEX P f g* **and** *LEX P g f*
  then obtain *i j* **where** *P* (*f i*) (*g i*) **and** *P* (*g j*) (*f j*)
    and $\forall k{<}i.\ f\ k = g\ k$ **and** $\forall k{<}j.\ g\ k = f\ k$ **by** (*auto simp*: *LEX-def*)
  then have *P* (*f* (*min i j*)) (*f* (*min i j*))
   using *assms*(*2*) **and** *SEQ* **by** (*cases* $i = j$) (*auto simp*: *antisymp-on-def min-def*,
force)
  with *assms*(*1*) **and** *SEQ* **show** $f = g$ **by** (*auto simp*: *irreflp-on-def*)
qed

lemma *LEX-trans*:
  assumes *transp-on A P* **and** $f \in SEQ\ A$ **and** $g \in SEQ\ A$ **and** $h \in SEQ\ A$
    and *LEX P f g* **and** *LEX P g h*
  shows *LEX P f h*
using *assms* **by** (*auto simp*: *LEX-def transp-on-def*) (*metis less-trans linorder-neqE-nat*)

lemma *qo-on-LEXEQ*:
  transp-on A P $\Longrightarrow$ *qo-on* (*LEXEQ P*) (*SEQ A*)
by (*auto simp*: *qo-on-def reflp-on-def transp-on-def* [*of - LEXEQ P*] *dest*: *LEX-trans*)

context *minimal-element*
begin

lemma *glb-LEX-lexmin*:
  assumes *chain-on* (*LEX P*) *C* (*SEQ A*) **and** $C \neq \{\}$
  shows *glb* (*LEX P*) *C* (*lexmin C*)
proof
  have $C \subseteq SEQ\ A$ **using** *assms* **by** (*auto simp*: *chain-on-def*)
  then have *lexmin C* $\in SEQ\ A$ **using** ‹$C \neq \{\}$› **by** (*intro lexmin-SEQ-mem*)
  note $* =$ ‹$C \subseteq SEQ\ A$› ‹$C \neq \{\}$›
  note *lex* = *LEX-imp-less* [*folded irreflp-on-def*, *OF po* [*THEN po-on-imp-irreflp-on*]]
  — *lexmin C* is a lower bound
  show *lb* (*LEX P*) *C* (*lexmin C*)
  proof
    fix *f* **assume** $f \in C$
    then show *LEXEQ P* (*lexmin C*) *f*
    proof (*cases* $f =$ *lexmin C*)
      define *i* **where** $i = (LEAST\ i.\ f\ i \neq lexmin\ C\ i)$
      case *False*
      then have *neq*: $\exists\, i.\ f\ i \neq lexmin\ C\ i$ **by** *blast*
      from *LeastI-ex* [*OF this*, *folded i-def*]

and *not-less-Least* [**where** $P = \lambda i.\ f\ i \neq lexmin\ C\ i$, *folded i-def*]
have *neq*: $f\ i \neq lexmin\ C\ i$ **and** *eq*: $\forall j{<}i.\ f\ j = lexmin\ C\ j$ **by** *auto*
then have $**$: $f \in eq\text{-}upto\ C\ (lexmin\ C)\ i\ f\ i \in ith\ (eq\text{-}upto\ C\ (lexmin\ C)\ i)$
i
using ‹$f \in C$› **by** *force+*
moreover from $**$ **have** $\neg\ P\ (f\ i)\ (lexmin\ C\ i)$
using *lexmin-minimal* [*OF* $*$, *of f i i*] **and** ‹$f \in C$› **and** ‹$C \subseteq SEQ\ A$› **by**
blast
moreover obtain g **where** $g \in eq\text{-}upto\ C\ (lexmin\ C)\ (Suc\ i)$
using *eq-upto-lexmin-non-empty* [*OF* $*$] **by** *blast*
ultimately have $P\ (lexmin\ C\ i)\ (f\ i)$
using *neq* **and** ‹$C \subseteq SEQ\ A$› **and** *assms*(*1*) **and** *lex* [*of g f i*] **and** *lex* [*of f*
g i]
by (*auto simp*: *eq-upto-def chain-on-def*)
with *eq* **show** *?thesis* **by** (*auto simp*: *LEX-def*)
qed *simp*
qed

— *lexmin C* is greater than or equal to any other lower bound
fix f **assume** *lb*: *lb* (*LEX P*) *C f*
then show *LEXEQ P f* (*lexmin C*)
proof (*cases* $f = lexmin\ C$)
define i **where** $i = (LEAST\ i.\ f\ i \neq lexmin\ C\ i)$
case *False*
then have *neq*: $\exists i.\ f\ i \neq lexmin\ C\ i$ **by** *blast*
from *LeastI-ex* [*OF this*, *folded i-def*]
and *not-less-Least* [**where** $P = \lambda i.\ f\ i \neq lexmin\ C\ i$, *folded i-def*]
have *neq*: $f\ i \neq lexmin\ C\ i$ **and** *eq*: $\forall j{<}i.\ f\ j = lexmin\ C\ j$ **by** *auto*
obtain h **where** $h \in eq\text{-}upto\ C\ (lexmin\ C)\ (Suc\ i)$ **and** $h \in C$
using *eq-upto-lexmin-non-empty* [*OF* $*$] **by** (*auto simp*: *eq-upto-def*)
then have [*simp*]: $\bigwedge j.\ j < Suc\ i \Longrightarrow h\ j = lexmin\ C\ j$ **by** *auto*
with *lb* **and** ‹$h \in C$› **have** *LEX P f h* **using** *neq* **by** (*auto simp*: *lb-def*)
then have $P\ (f\ i)\ (h\ i)$
using *neq* **and** *eq* **and** ‹$C \subseteq SEQ\ A$› **and** ‹$h \in C$› **by** (*intro lex*) *auto*
with *eq* **show** *?thesis* **by** (*auto simp*: *LEX-def*)
qed *simp*
qed

lemma *dc-on-LEXEQ*:
dc-on (*LEXEQ P*) (*SEQ A*)
proof
fix C **assume** *chain-on* (*LEXEQ P*) *C* (*SEQ A*) **and** $C \neq \{\}$
then have *chain*: *chain-on* (*LEX P*) *C* (*SEQ A*) **by** (*auto simp*: *chain-on-def*)
then have $C \subseteq SEQ\ A$ **by** (*auto simp*: *chain-on-def*)
then have $lexmin\ C \in SEQ\ A$ **using** ‹$C \neq \{\}$› **by** (*intro lexmin-SEQ-mem*)
have *glb* (*LEX P*) *C* (*lexmin C*) **by** (*rule glb-LEX-lexmin* [*OF chain* ‹$C \neq \{\}$›])
then have *glb* (*LEXEQ P*) *C* (*lexmin C*) **by** (*auto simp*: *glb-def lb-def*)
with ‹$lexmin\ C \in SEQ\ A$› **show** $\exists f \in SEQ\ A.\ glb\ (LEXEQ\ P)\ C\ f$ **by** *blast*
qed

end

Properties that only depend on finite initial segments of a sequence (i.e., which are open with respect to the product topology).

definition *pt-open-on* Q A $\longleftrightarrow$ $(\forall f{\in}A.\ Q\ f \longleftrightarrow (\exists\, n.\ (\forall\, g{\in}A.\ (\forall\, i{<}n.\ g\ i = f\ i) \longrightarrow Q\ g)))$

lemma *pt-open-onD*:
  $\textit{pt-open-on}\ Q\ A \Longrightarrow Q\ f \Longrightarrow f \in A \Longrightarrow (\exists\, n.\ (\forall\, g{\in}A.\ (\forall\, i{<}n.\ g\ i = f\ i) \longrightarrow Q\ g))$
  unfolding *pt-open-on-def* **by** *blast*

lemma *pt-open-on-good*:
  pt-open-on (*good Q*) (*SEQ A*)
proof (*unfold pt-open-on-def*, *intro ballI*)
  fix *f* **assume** *f*: $f \in \textit{SEQ}\ A$
  show $\textit{good}\ Q\ f = (\exists\, n.\ \forall\, g{\in}\textit{SEQ}\ A.\ (\forall\, i{<}n.\ g\ i = f\ i) \longrightarrow \textit{good}\ Q\ g)$
  proof
    assume *good Q f*
    then obtain *i* **and** *j* **where** $*$: $i < j$ $Q\ (f\ i)\ (f\ j)$ **by** *auto*
    have $\forall\, g{\in}\textit{SEQ}\ A.\ (\forall\, i{<}\textit{Suc}\ j.\ g\ i = f\ i) \longrightarrow \textit{good}\ Q\ g$
    proof (*intro ballI impI*)
      fix *g* **assume** $g \in \textit{SEQ}\ A$ **and** $\forall\, i{<}\textit{Suc}\ j.\ g\ i = f\ i$
      then show *good Q g* **using** $*$ **by** (*force simp*: *good-def*)
    qed
    then show $\exists\, n.\ \forall\, g{\in}\textit{SEQ}\ A.\ (\forall\, i{<}n.\ g\ i = f\ i) \longrightarrow \textit{good}\ Q\ g$ **..**
  next
    assume $\exists\, n.\ \forall\, g{\in}\textit{SEQ}\ A.\ (\forall\, i{<}n.\ g\ i = f\ i) \longrightarrow \textit{good}\ Q\ g$
    with *f* **show** *good Q f* **by** *blast*
  qed
qed

context *minimal-element*
begin

lemma *pt-open-on-imp-open-on-LEXEQ*:
  assumes *pt-open-on Q* (*SEQ A*)
  shows *open-on* (*LEXEQ P*) *Q* (*SEQ A*)
proof
  fix *C* **assume** *chain*: *chain-on* (*LEXEQ P*) *C* (*SEQ A*) **and** *ne*: $C \neq \{\}$
    and $\exists\, g{\in}\textit{SEQ}\ A.\ \textit{glb}\ (\textit{LEXEQ}\ P)\ C\ g \wedge Q\ g$
  then obtain *g* **where** *g*: $g \in \textit{SEQ}\ A$ **and** *glb* (*LEXEQ P*) *C g*
    and *Q*: *Q g* **by** *blast*
  then have *glb*: *glb* (*LEX P*) *C g* **by** (*auto simp*: *glb-def lb-def*)
  from *chain* **have** *chain-on* (*LEX P*) *C* (*SEQ A*) **and** *C*: $C \subseteq \textit{SEQ}\ A$ **by** (*auto simp*: *chain-on-def*)
  note $*$ = *glb-LEX-lexmin* [*OF this*(*1*) *ne*]
  have $\textit{lexmin}\ C \in \textit{SEQ}\ A$ **using** *ne* **and** *C* **by** (*intro lexmin-SEQ-mem*)

from *glb-unique* [*OF* - *g this glb* ∗]
and *antisymp-on-LEX* [*OF po-on-imp-irreflp-on* [*OF po*] *po-on-imp-antisymp-on* [*OF po*]]
have [*simp*]: *lexmin C* = *g* **by** *auto*
from *assms* [*THEN pt-open-onD*, *OF Q g*]
obtain *n* :: *nat* **where** ∗∗: $\bigwedge h.\ h \in SEQ\ A \Longrightarrow (\forall i{<}n.\ h\ i = g\ i) \longrightarrow Q\ h$ **by** *blast*
from *eq-upto-lexmin-non-empty* [*OF C ne*, *of n*]
obtain *f* **where** *f* ∈ *eq-upto C g n* **by** *auto*
then have *f* ∈ *C* **and** *Q f* **using** ∗∗ [*of f*] **and** *C* **by** *force*+
then show $\exists f{\in}C.\ Q\ f$ **by** *blast*
qed

lemma *open-on-good*:
open-on (*LEXEQ P*) (*good Q*) (*SEQ A*)
by (*intro pt-open-on-imp-open-on-LEXEQ pt-open-on-good*)

end

lemma *open-on-LEXEQ-imp-pt-open-on-counterexample*:
fixes *a b* :: ′*a*
defines $A \equiv \{a, b\}$ **and** $P \equiv (\lambda x\ y.\ False)$ **and** $Q \equiv (\lambda f.\ \forall i.\ f\ i = b)$
assumes [*simp*]: $a \neq b$
shows *minimal-element P A* **and** *open-on* (*LEXEQ P*) *Q* (*SEQ A*)
and ¬ *pt-open-on Q* (*SEQ A*)
proof −
show *minimal-element P A*
by *standard* (*auto simp*: *P-def po-on-def irreflp-on-def transp-on-def wfp-on-def*)
show *open-on* (*LEXEQ P*) *Q* (*SEQ A*)
by (*auto simp*: *P-def open-on-def chain-on-def SEQ-def glb-def lb-def LEX-def*)
show ¬ *pt-open-on Q* (*SEQ A*)
proof
define *f* :: *nat* ⇒ ′*a* **where** $f \equiv (\lambda x.\ b)$
have *f* ∈ *SEQ A* **by** (*auto simp*: *A-def f-def*)
moreover assume *pt-open-on Q* (*SEQ A*)
ultimately have $Q\ f \longleftrightarrow (\exists n.\ (\forall g{\in}SEQ\ A.\ (\forall i{<}n.\ g\ i = f\ i) \longrightarrow Q\ g))$
unfolding *pt-open-on-def* **by** *blast*
moreover have *Q f* **by** (*auto simp*: *Q-def f-def*)
moreover have $\exists g{\in}SEQ\ A.\ (\forall i{<}n.\ g\ i = f\ i) \land \neg\ Q\ g$ **for** *n*
by (*intro bexI* [*of* - *f*(*n* := *a*)]) (*auto simp*: *f-def Q-def A-def*)
ultimately show *False* **by** *blast*
qed
qed

lemma *higman*:
assumes *almost-full-on P A*
shows *almost-full-on* (*list-emb P*) (*lists A*)
proof
interpret *minimal-element strict-suffix lists A*

by (*unfold-locales*) (*intro po-on-strict-suffix wfp-on-strict-suffix*)+
fix *f* **presume** $f \in SEQ$ (*lists A*)
with *qo-on-LEXEQ* [*OF po-on-imp-transp-on* [*OF po-on-strict-suffix*]] **and** *dc-on-LEXEQ* **and** *open-on-good*
show *good* (*list-emb P*) *f*
proof (*induct rule*: *open-induct-on*)
case (*less f*)
define *h* **where** $h\ i = hd\ (f\ i)$ **for** *i*
show *?case*
proof (*cases* $\exists\, i.\ f\ i = []$)
case *False*
then have *ne*: $\forall\, i.\ f\ i \neq []$ **by** *auto*
with ‹$f \in SEQ$ (*lists A*)› **have** $\forall\, i.\ h\ i \in A$ **by** (*auto simp*: *h-def ne-lists*)
from *almost-full-on-imp-homogeneous-subseq* [*OF assms this*]
obtain $\varphi :: nat \Rightarrow nat$ **where** *mono*: $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$
and *P*: $\bigwedge i\ j.\ i < j \Longrightarrow P\ (h\ (\varphi\ i))\ (h\ (\varphi\ j))$ **by** *blast*
define f' **where** $f'\ i = (if\ i < \varphi\ 0\ then\ f\ i\ else\ tl\ (f\ (\varphi\ (i - \varphi\ 0))))$ **for** *i*
have f': $f' \in SEQ$ (*lists A*) **using** *ne* **and** ‹$f \in SEQ$ (*lists A*)›
by (*auto simp*: f'*-def dest*: *list.set-sel*)
have [*simp*]: $\bigwedge i.\ \varphi\ 0 \leq i \Longrightarrow h\ (\varphi\ (i - \varphi\ 0))\ \#\ f'\ i = f\ (\varphi\ (i - \varphi\ 0))$
$\bigwedge i.\ i < \varphi\ 0 \Longrightarrow f'\ i = f\ i$ **using** *ne* **by** (*auto simp*: f'*-def h-def*)
moreover have *strict-suffix* $(f'\ (\varphi\ 0))\ (f\ (\varphi\ 0))$ **using** *ne* **by** (*auto simp*: f'*-def*)
ultimately have *LEX strict-suffix* $f'\ f$ **by** (*auto simp*: *LEX-def*)
with *LEX-imp-not-LEX* [*OF this*] **have** *strict* (*LEXEQ strict-suffix*) $f'\ f$
using *po-on-strict-suffix* [*of UNIV*] **unfolding** *po-on-def irreflp-on-def transp-on-def* **by** *blast*
from *less*(*2*) [*OF* f' *this*] **have** *good* (*list-emb P*) f' **.**
then obtain *i j* **where** $i < j$ **and** *emb*: *list-emb P* $(f'\ i)\ (f'\ j)$ **by** (*auto simp*: *good-def*)
consider $j < \varphi\ 0 \mid \varphi\ 0 \leq i \mid i < \varphi\ 0$ **and** $\varphi\ 0 \leq j$ **by** *arith*
then show *?thesis*
proof (*cases*)
case *1* **with** ‹$i < j$› **and** *emb* **show** *?thesis* **by** (*auto simp*: *good-def*)
next
case *2*
with ‹$i < j$› **and** *P* **have** $P\ (h\ (\varphi\ (i - \varphi\ 0)))\ (h\ (\varphi\ (j - \varphi\ 0)))$ **by** *auto*
with *emb* **have** *list-emb P* $(h\ (\varphi\ (i - \varphi\ 0))\ \#\ f'\ i)\ (h\ (\varphi\ (j - \varphi\ 0))\ \#\ f'\ j)$ **by** *auto*
then have *list-emb P* $(f\ (\varphi\ (i - \varphi\ 0)))\ (f\ (\varphi\ (j - \varphi\ 0)))$ **using** *2* **and** ‹$i < j$› **by** *auto*
moreover with *2* **and** ‹$i < j$› **have** $\varphi\ (i - \varphi\ 0) < \varphi\ (j - \varphi\ 0)$ **using** *mono* **by** *auto*
ultimately show *?thesis* **by** (*auto simp*: *good-def*)
next
case *3*
with *emb* **have** *list-emb P* $(f\ i)\ (f'\ j)$ **by** *auto*
moreover have $f\ (\varphi\ (j - \varphi\ 0)) = h\ (\varphi\ (j - \varphi\ 0))\ \#\ f'\ j$ **using** *3* **by** *auto*
ultimately have *list-emb P* $(f\ i)\ (f\ (\varphi\ (j - \varphi\ 0)))$ **by** *auto*

moreover have $i < \varphi\ (j - \varphi\ 0)$ **using** *mono* $[of\ 0\ j - \varphi\ 0]$ **and** *3* **by** *force*
ultimately show *?thesis* **by** (*auto simp*: *good-def*)
qed
qed *auto*
qed
qed *blast*

end

7 Almost-Full Relations

theory *Almost-Full-Relations*
imports *Minimal-Bad-Sequences*
begin

lemma (**in** *mbs*) *mbs′*:
assumes $\neg$ *almost-full-on P A*
shows $\exists\, m \in BAD\ P.\ \forall\, g.\ (m,\ g) \in gseq \longrightarrow good\ P\ g$
using *assms* **and** *mbs* **unfolding** *almost-full-on-def* **by** *blast*

7.1 Adding a Bottom Element to a Set

definition *with-bot* :: $'a\ set \Rightarrow 'a\ option\ set$ (‹-$_\bot$› [*1000*] *1000*)
where
$A_\bot = \{None\} \cup Some\ `\ A$

lemma *with-bot-iff* [*iff*]:
$Some\ x \in A_\bot \longleftrightarrow x \in A$
by (*auto simp*: *with-bot-def*)

lemma *NoneI* [*simp*, *intro*]:
$None \in A_\bot$
by (*simp add*: *with-bot-def*)

lemma *not-None-the-mem* [*simp*]:
$x \neq None \Longrightarrow the\ x \in A \longleftrightarrow x \in A_\bot$
by *auto*

lemma *with-bot-cases*:
$u \in A_\bot \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow u = Some\ x \Longrightarrow P) \Longrightarrow (u = None \Longrightarrow P) \Longrightarrow P$
by *auto*

lemma *with-bot-empty-conv* [*iff*]:
$A_\bot = \{None\} \longleftrightarrow A = \{\}$
by (*auto elim*: *with-bot-cases*)

lemma *with-bot-UNIV* [*simp*]:
$UNIV_\bot = UNIV$
proof (*rule set-eqI*)

fix x :: $'a$ *option*
show $x \in UNIV_\bot \longleftrightarrow x \in UNIV$ **by** (*cases x*) *auto*
qed

7.2 Adding a Bottom Element to an Almost-Full Set

fun
option-le :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ option \Rightarrow 'a\ option \Rightarrow bool$
where
option-le P None y = True |
option-le P (*Some x*) *None = False* |
option-le P (*Some x*) (*Some y*) = *P x y*

lemma *None-imp-good-option-le* [*simp*]:
assumes *f i = None*
shows *good* (*option-le P*) *f*
by (*rule goodI* [*of i Suc i*]) (*auto simp*: *assms*)

lemma *almost-full-on-with-bot*:
assumes *almost-full-on P A*
shows *almost-full-on* (*option-le P*) $A_\bot$ (**is** *almost-full-on ?P ?A*)
proof
fix f :: $nat \Rightarrow 'a\ option$
assume $*$: $\forall i.\ f\ i \in\ ?A$
show *good ?P f*
proof (*cases* $\forall i.\ f\ i \neq None$)
case *True*
then have $**$: $\bigwedge i.\ Some\ (the\ (f\ i)) = f\ i$
and $\bigwedge i.\ the\ (f\ i) \in A$ **using** $*$ **by** *auto*
with *almost-full-onD* [*OF assms*, *of the* $\circ$ *f*] **obtain** *i j* **where** $i < j$
and *P* (*the* (*f i*)) (*the* (*f j*)) **by** *auto*
then have *?P* (*Some* (*the* (*f i*))) (*Some* (*the* (*f j*))) **by** *simp*
then have *?P* (*f i*) (*f j*) **unfolding** $**$ **.**
with ‹$i < j$› **show** *good ?P f* **by** (*auto simp*: *good-def*)
qed *auto*
qed

7.3 Disjoint Union of Almost-Full Sets

fun
sum-le :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a + 'b \Rightarrow 'a + 'b \Rightarrow bool$
where
sum-le P Q (*Inl x*) (*Inl y*) = *P x y* |
sum-le P Q (*Inr x*) (*Inr y*) = *Q x y* |
sum-le P Q x y = False

lemma *not-sum-le-cases*:
assumes $\neg$ *sum-le P Q a b*
and $\bigwedge x\ y.\ [\![a = Inl\ x;\ b = Inl\ y;\ \neg\ P\ x\ y]\!] \Longrightarrow thesis$
and $\bigwedge x\ y.\ [\![a = Inr\ x;\ b = Inr\ y;\ \neg\ Q\ x\ y]\!] \Longrightarrow thesis$

and $\bigwedge x\ y.\ [\![a = Inl\ x;\ b = Inr\ y]\!] \Longrightarrow thesis$
and $\bigwedge x\ y.\ [\![a = Inr\ x;\ b = Inl\ y]\!] \Longrightarrow thesis$
shows *thesis*
using *assms* **by** (*cases a b rule*: *sum.exhaust* [*case-product sum.exhaust*]) *auto*

When two sets are almost-full, then their disjoint sum is almost-full.

lemma *almost-full-on-Plus*:
assumes *almost-full-on P A* **and** *almost-full-on Q B*
shows *almost-full-on* (*sum-le P Q*) (*A* <+> *B*) (**is** *almost-full-on ?P ?A*)
proof
fix $f :: nat \Rightarrow ('a + 'b)$
let $?I = f\ -`\ Inl\ `\ A$
let $?J = f\ -`\ Inr\ `\ B$
assume $\forall i.\ f\ i \in\ ?A$
then have $*$: $?J = (UNIV::nat\ set) -\ ?I$ **by** (*fastforce*)
show *good ?P f*
proof (*rule ccontr*)
assume *bad*: *bad ?P f*
show *False*
proof (*cases finite ?I*)
assume *finite ?I*
then have *infinite ?J* **by** (*auto simp*: $*$)
then interpret *infinitely-many1* $\lambda i.\ f\ i \in Inr\ `\ B$
by (*unfold-locales*) (*simp add*: *infinite-nat-iff-unbounded*)
have [*dest*]: $\bigwedge i\ x.\ f\ (enum\ i) = Inl\ x \Longrightarrow False$
using *enum-P* **by** (*auto simp*: *image-iff*) (*metis Inr-Inl-False*)
let $?f = \lambda i.\ projr\ (f\ (enum\ i))$
have *B*: $\bigwedge i.\ ?f\ i \in B$ **using** *enum-P* **by** (*auto simp*: *image-iff*) (*metis sum.sel*(*2*))
{ fix $i\ j :: nat$
assume $i < j$
then have $enum\ i < enum\ j$ **using** *enum-less* **by** *auto*
with *bad* **have** $\neg\ ?P\ (f\ (enum\ i))\ (f\ (enum\ j))$ **by** (*auto simp*: *good-def*)
then have $\neg\ Q\ (?f\ i)\ (?f\ j)$ **by** (*auto elim*: *not-sum-le-cases*) **}**
then have *bad Q ?f* **by** (*auto simp*: *good-def*)
moreover from ‹*almost-full-on Q B*› **and** *B*
have *good Q ?f* **by** (*auto simp*: *good-def almost-full-on-def*)
ultimately show *False* **by** *blast*
next
assume *infinite ?I*
then interpret *infinitely-many1* $\lambda i.\ f\ i \in Inl\ `\ A$
by (*unfold-locales*) (*simp add*: *infinite-nat-iff-unbounded*)
have [*dest*]: $\bigwedge i\ x.\ f\ (enum\ i) = Inr\ x \Longrightarrow False$
using *enum-P* **by** (*auto simp*: *image-iff*) (*metis Inr-Inl-False*)
let $?f = \lambda i.\ projl\ (f\ (enum\ i))$
have *A*: $\forall i.\ ?f\ i \in A$ **using** *enum-P* **by** (*auto simp*: *image-iff*) (*metis sum.sel*(*1*))
{ fix $i\ j :: nat$
assume $i < j$

then have *enum i* $<$ *enum j* **using** *enum-less* **by** *auto*
with *bad* **have** $\neg$ *?P* (*f* (*enum i*)) (*f* (*enum j*)) **by** (*auto simp*: *good-def*)
then have $\neg$ *P* (*?f i*) (*?f j*) **by** (*auto elim*: *not-sum-le-cases*) **}**
then have *bad P ?f* **by** (*auto simp*: *good-def*)
moreover from ‹*almost-full-on P A*› **and** *A*
have *good P ?f* **by** (*auto simp*: *good-def almost-full-on-def*)
ultimately show *False* **by** *blast*
qed
qed
qed

7.4 Dickson's Lemma for Almost-Full Relations

When two sets are almost-full, then their Cartesian product is almost-full.

definition
prod-le :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a \times 'b \Rightarrow 'a \times 'b \Rightarrow bool$
where
prod-le P1 P2 $= (\lambda(p1, p2)\ (q1, q2).\ P1\ p1\ q1 \wedge P2\ p2\ q2)$

lemma *prod-le-True* [*simp*]:
prod-le P $(\lambda\text{- -}.\ True)\ a\ b = P\ (fst\ a)\ (fst\ b)$
by (*auto simp*: *prod-le-def*)

lemma *almost-full-on-Sigma*:
assumes *almost-full-on P1 A1* **and** *almost-full-on P2 A2*
shows *almost-full-on* (*prod-le P1 P2*) $(A1 \times A2)$ (**is** *almost-full-on ?P ?A*)
proof (*rule ccontr*)
assume $\neg$ *almost-full-on ?P ?A*
then obtain *f* **where** *f*: $\forall i.\ f\ i \in ?A$
and *bad*: *bad ?P f* **by** (*auto simp*: *almost-full-on-def*)
let $?W = \lambda x\ y.\ P1\ (fst\ x)\ (fst\ y)$
let $?B = \lambda x\ y.\ P2\ (snd\ x)\ (snd\ y)$
from *f* **have** *fst*: $\forall i.\ fst\ (f\ i) \in A1$ **and** *snd*: $\forall i.\ snd\ (f\ i) \in A2$
by (*metis SigmaE fst-conv*, *metis SigmaE snd-conv*)
from *almost-full-on-imp-homogeneous-subseq* [*OF assms*(*1*) *fst*]
obtain $\varphi :: nat \Rightarrow nat$ **where** *mono*: $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$
and $*$: $\bigwedge i\ j.\ i < j \Longrightarrow\ ?W\ (f\ (\varphi\ i))\ (f\ (\varphi\ j))$ **by** *auto*
from *snd* **have** $\forall i.\ snd\ (f\ (\varphi\ i)) \in A2$ **by** *auto*
then have $snd \circ f \circ \varphi \in SEQ\ A2$ **by** *auto*
with *assms*(*2*) **have** *good P2* $(snd \circ f \circ \varphi)$ **by** (*auto simp*: *almost-full-on-def*)
then obtain *i j* :: *nat*
where $i < j$ **and** $?B\ (f\ (\varphi\ i))\ (f\ (\varphi\ j))$ **by** *auto*
with $*$ [*OF* ‹$i < j$›] **have** $?P\ (f\ (\varphi\ i))\ (f\ (\varphi\ j))$ **by** (*simp add*: *case-prod-beta prod-le-def*)
with *mono* [*OF* ‹$i < j$›] **and** *bad* **show** *False* **by** *auto*
qed

7.5 Higman's Lemma for Almost-Full Relations

lemma *almost-full-on-lists*:
  assumes *almost-full-on P A*
  shows *almost-full-on* (*list-emb P*) (*lists A*) (**is** *almost-full-on ?P ?A*)
proof (*rule ccontr*)
  interpret *mbs ?A* **.**
  assume ¬ *?thesis*
  from *mbs′* [*OF this*] **obtain** *m*
    where *bad*: $m \in BAD\ ?P$
    and *min*: $\forall g.\ (m, g) \in gseq \longrightarrow good\ ?P\ g$ **..**
  then have *lists*: $\bigwedge i.\ m\ i \in lists\ A$
    and *ne*: $\bigwedge i.\ m\ i \neq []$ **by** *auto*

  define *h t* **where** $h = (\lambda i.\ hd\ (m\ i))$ **and** $t = (\lambda i.\ tl\ (m\ i))$
  have *m*: $\bigwedge i.\ m\ i = h\ i\ \#\ t\ i$ **using** *ne* **by** (*simp add*: *h-def t-def*)

  have $\forall i.\ h\ i \in A$ **using** *ne-lists* [*OF ne*] **and** *lists* **by** (*auto simp add*: *h-def*)
  from *almost-full-on-imp-homogeneous-subseq* [*OF assms this*] **obtain** $\varphi :: nat \Rightarrow nat$
    where *less*: $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$
      and *P*: $\forall i\ j.\ i < j \longrightarrow P\ (h\ (\varphi\ i))\ (h\ (\varphi\ j))$ **by** *blast*

  have *bad-t*: $bad\ ?P\ (t \circ \varphi)$
  proof
    assume $good\ ?P\ (t \circ \varphi)$
    then obtain *i j* **where** $i < j$ **and** $?P\ (t\ (\varphi\ i))\ (t\ (\varphi\ j))$ **by** *auto*
    moreover with *P* **have** $P\ (h\ (\varphi\ i))\ (h\ (\varphi\ j))$ **by** *blast*
    ultimately have $?P\ (m\ (\varphi\ i))\ (m\ (\varphi\ j))$
      by (*subst* (*1 2*) *m*) (*rule list-emb-Cons2*, *auto*)
    with *less* **and** ‹$i < j$› **have** *good ?P m* **by** (*auto simp*: *good-def*)
    with *bad* **show** *False* **by** *blast*
  qed

  define *m′* **where** $m' = (\lambda i.\ if\ i < \varphi\ 0\ then\ m\ i\ else\ t\ (\varphi\ (i - \varphi\ 0)))$

  have *m′-less*: $\bigwedge i.\ i < \varphi\ 0 \Longrightarrow m'\ i = m\ i$ **by** (*simp add*: *m′-def*)
  have *m′-geq*: $\bigwedge i.\ i \geq \varphi\ 0 \Longrightarrow m'\ i = t\ (\varphi\ (i - \varphi\ 0))$ **by** (*simp add*: *m′-def*)

  have $\forall i.\ m'\ i \in lists\ A$ **using** *ne-lists* [*OF ne*] **and** *lists* **by** (*auto simp*: *m′-def t-def*)
  moreover have $length\ (m'\ (\varphi\ 0)) < length\ (m\ (\varphi\ 0))$ **using** *ne* **by** (*simp add*: *t-def m′-geq*)
  moreover have $\forall j<\varphi\ 0.\ m'\ j = m\ j$ **by** (*auto simp*: *m′-less*)
  ultimately have $(m, m') \in gseq$ **using** *lists* **by** (*auto simp*: *gseq-def*)
  moreover have *bad ?P m′*
  proof
    assume *good ?P m′*
    then obtain *i j* **where** $i < j$ **and** *emb*: $?P\ (m'\ i)\ (m'\ j)$ **by** (*auto simp*: *good-def*)

```
  { assume j < φ 0
    with ‹i < j› and emb have ?P (m i) (m j) by (auto simp: m'-less)
    with ‹i < j› and bad have False by blast }
  moreover
  { assume φ 0 ≤ i
    with ‹i < j› and emb have ?P (t (φ (i − φ 0))) (t (φ (j − φ 0)))
      and i − φ 0 < j − φ 0 by (auto simp: m'-geq)
    with bad-t have False by auto }
  moreover
  { assume i < φ 0 and φ 0 ≤ j
   with ‹i < j› and emb have ?P (m i) (t (φ (j − φ 0))) by (simp add: m'-less
m'-geq)
    from list-emb-Cons [OF this, of h (φ (j − φ 0))]
      have ?P (m i) (m (φ (j − φ 0))) using ne by (simp add: h-def t-def)
    moreover have i < φ (j − φ 0)
      using less [of 0 j − φ 0] and ‹i < φ 0› and ‹φ 0 ≤ j›
      by (cases j = φ 0) auto
    ultimately have False using bad by blast }
  ultimately show False using ‹i < j› by arith
 qed
 ultimately show False using min by blast
qed
```

7.6 Natural Numbers

```
lemma almost-full-on-UNIV-nat:
 almost-full-on (≤) (UNIV :: nat set)
proof −
 let ?P = subseq :: bool list ⇒ bool list ⇒ bool
 have ∗: length ‘ (UNIV :: bool list set) = (UNIV :: nat set)
  by (metis Ex-list-of-length surj-def)
 have almost-full-on (≤) (length ‘ (UNIV :: bool list set))
 proof (rule almost-full-on-hom)
  fix xs ys :: bool list
  assume ?P xs ys
  then show length xs ≤ length ys
   by (metis list-emb-length)
 next
  have finite (UNIV :: bool set) by auto
  from almost-full-on-lists [OF eq-almost-full-on-finite-set [OF this]]
   show almost-full-on ?P UNIV unfolding lists-UNIV .
 qed
 then show ?thesis unfolding ∗ .
qed

end
```

8 Well-Quasi-Orders

theory *Well-Quasi-Orders*
imports *Almost-Full-Relations*
begin

8.1 Basic Definitions

definition *wqo-on* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$ **where**
wqo-on P A $\longleftrightarrow$ *transp-on A P* $\wedge$ *almost-full-on P A*

lemma *wqo-on-UNIV*:
wqo-on (λ- -. *True*) *UNIV*
using *almost-full-on-UNIV* **by** (*auto simp*: *wqo-on-def transp-on-def*)

lemma *wqo-onI* [*Pure.intro*]:
$[\![$*transp-on A P*; *almost-full-on P A*$]\!] \Longrightarrow$ *wqo-on P A*
unfolding *wqo-on-def almost-full-on-def* **by** *blast*

lemma *wqo-on-imp-reflp-on*:
wqo-on P A $\Longrightarrow$ *reflp-on A P*
using *almost-full-on-imp-reflp-on* **by** (*auto simp*: *wqo-on-def*)

lemma *wqo-on-imp-transp-on*:
wqo-on P A $\Longrightarrow$ *transp-on A P*
by (*auto simp*: *wqo-on-def*)

lemma *wqo-on-imp-almost-full-on*:
wqo-on P A $\Longrightarrow$ *almost-full-on P A*
by (*auto simp*: *wqo-on-def*)

lemma *wqo-on-imp-qo-on*:
wqo-on P A $\Longrightarrow$ *qo-on P A*
by (*metis qo-on-def wqo-on-imp-reflp-on wqo-on-imp-transp-on*)

lemma *wqo-on-imp-good*:
wqo-on P A $\Longrightarrow \forall\, i.\ f\ i \in A \Longrightarrow$ *good P f*
by (*auto simp*: *wqo-on-def almost-full-on-def*)

lemma *wqo-on-subset*:
$A \subseteq B \Longrightarrow$ *wqo-on P B* $\Longrightarrow$ *wqo-on P A*
using *almost-full-on-subset* [*of A B P*]
and *transp-on-subset* [*of B P A*]
unfolding *wqo-on-def* **by** *blast*

8.2 Equivalent Definitions

Given a quasi-order P, the following statements are equivalent:

1. P is a almost-full.

2. P does neither allow decreasing chains nor antichains.

3. Every quasi-order extending P is well-founded.

lemma *wqo-af-conv*:
  assumes *qo-on P A*
  shows *wqo-on P A* $\longleftrightarrow$ *almost-full-on P A*
  using *assms* **by** (*metis qo-on-def wqo-on-def*)

lemma *wqo-wf-and-no-antichain-conv*:
  assumes *qo-on P A*
  shows *wqo-on P A* $\longleftrightarrow$ *wfp-on* (*strict P*) $A \wedge \neg$ ($\exists f$. *antichain-on P f A*)
  unfolding *wqo-af-conv* [*OF assms*]
  using *af-trans-imp-wf* [*OF - assms* [*THEN qo-on-imp-transp-on*]]
    and *almost-full-on-imp-no-antichain-on* [*of P A*]
    and *wf-and-no-antichain-imp-qo-extension-wf* [*of P A*]
    and *every-qo-extension-wf-imp-af* [*OF - assms*]
    by *blast*

lemma *wqo-extensions-wf-conv*:
  assumes *qo-on P A*
  shows *wqo-on P A* $\longleftrightarrow$ ($\forall Q$. ($\forall x \in A$. $\forall y \in A$. $P\ x\ y \longrightarrow Q\ x\ y$) $\wedge$ *qo-on Q A* $\longrightarrow$ *wfp-on* (*strict Q*) A)
  unfolding *wqo-af-conv* [*OF assms*]
  using *af-trans-imp-wf* [*OF - assms* [*THEN qo-on-imp-transp-on*]]
    and *almost-full-on-imp-no-antichain-on* [*of P A*]
    and *wf-and-no-antichain-imp-qo-extension-wf* [*of P A*]
    and *every-qo-extension-wf-imp-af* [*OF - assms*]
    by *blast*

lemma *wqo-on-imp-wfp-on*:
  wqo-on P A $\Longrightarrow$ *wfp-on* (*strict P*) A
  by (*metis* (*no-types*) *wqo-on-imp-qo-on wqo-wf-and-no-antichain-conv*)

The homomorphic image of a wqo set is wqo.

lemma *wqo-on-hom*:
  assumes *transp-on* (h ‘ A) Q
    and $\forall x \in A$. $\forall y \in A$. $P\ x\ y \longrightarrow Q\ (h\ x)\ (h\ y)$
    and *wqo-on P A*
  shows *wqo-on Q* (h ‘ A)
  using *assms* **and** *almost-full-on-hom* [*of A P Q h*]
  unfolding *wqo-on-def* **by** *blast*

The monomorphic preimage of a wqo set is wqo.

lemma *wqo-on-mon*:
  assumes $*$: $\forall x \in A$. $\forall y \in A$. $P\ x\ y \longleftrightarrow Q\ (h\ x)\ (h\ y)$
    and *bij*: *bij-betw h A B*
    and *wqo*: *wqo-on Q B*
  shows *wqo-on P A*

proof −
  have *transp-on A P*
  proof (*rule transp-onI*)
    fix *x y z* **assume** [*intro*!]: *x* ∈ *A y* ∈ *A z* ∈ *A*
      and *P x y* **and** *P y z*
    with ∗ **have** *Q* (*h x*) (*h y*) **and** *Q* (*h y*) (*h z*) **by** *blast*+
    with *wqo-on-imp-transp-on* [*OF wqo*] **have** *Q* (*h x*) (*h z*)
      using *bij* **by** (*auto simp*: *bij-betw-def transp-on-def*)
    with ∗ **show** *P x z* **by** *blast*
  qed
  with *assms* **and** *almost-full-on-mon* [*of A P Q h*]
    show *?thesis* **unfolding** *wqo-on-def* **by** *blast*
qed

8.3 A Type Class for Well-Quasi-Orders

In a well-quasi-order (wqo) every infinite sequence is good.

class *wqo* = *preorder* +
  assumes *good*: *good* (≤) *f*

lemma *wqo-on-class* [*simp*, *intro*]:
  wqo-on (≤) (*UNIV* :: (*'a* :: *wqo*) *set*)
  using *good* **by** (*auto simp*: *wqo-on-def transp-on-def almost-full-on-def dest*: *order-trans*)

lemma *wqo-on-UNIV-class-wqo* [*intro*!]:
  wqo-on P UNIV ⟹ *class.wqo P* (*strict P*)
  by (*unfold-locales*) (*auto simp*: *wqo-on-def almost-full-on-def*, *unfold transp-on-def*, *blast*)

The following lemma converts between *wqo-on* (for the special case that the domain is the universe of a type) and the class predicate *class.wqo*.

lemma *wqo-on-UNIV-conv*:
  wqo-on P UNIV ⟷ *class.wqo P* (*strict P*) (**is** *?lhs* = *?rhs*)
proof
  assume *?lhs* **then show** *?rhs* **by** *auto*
next
  assume *?rhs* **then show** *?lhs*
    unfolding *class.wqo-def class.preorder-def class.wqo-axioms-def*
    by (*auto simp*: *wqo-on-def almost-full-on-def transp-on-def*)
qed

The strict part of a wqo is well-founded.

lemma (**in** *wqo*) *wfP* (<)
proof −
  have *class.wqo* (≤) (<) **..**
  hence *wqo-on* (≤) *UNIV*
    unfolding *less-le-not-le* [*abs-def*] *wqo-on-UNIV-conv* [*symmetric*] **.**

from *wqo-on-imp-wfp-on* [*OF this*]
  show *?thesis* **unfolding** *less-le-not-le* [*abs-def*] *wfp-on-UNIV* **.**
qed

lemma *wqo-on-with-bot*:
  assumes *wqo-on P A*
  shows *wqo-on* (*option-le P*) $A_\bot$ (**is** *wqo-on ?P ?A*)
proof −
  { from *assms* **have** *trans* [*unfolded transp-on-def*]: *transp-on A P*
      by (*auto simp*: *wqo-on-def*)
    have *transp-on ?A ?P*
      by (*auto simp*: *transp-on-def elim*!: *with-bot-cases*, *insert trans*) *blast* **}**
  moreover
  { from *assms* **and** *almost-full-on-with-bot*
      have *almost-full-on ?P ?A* **by** (*auto simp*: *wqo-on-def*) **}**
  ultimately
  show *?thesis* **by** (*auto simp*: *wqo-on-def*)
qed

lemma *wqo-on-option-UNIV* [*intro*]:
  wqo-on P UNIV $\Longrightarrow$ *wqo-on* (*option-le P*) *UNIV*
  using *wqo-on-with-bot* [*of P UNIV*] **by** *simp*

When two sets are wqo, then their disjoint sum is wqo.

lemma *wqo-on-Plus*:
  assumes *wqo-on P A* **and** *wqo-on Q B*
  shows *wqo-on* (*sum-le P Q*) (*A* <+> *B*) (**is** *wqo-on ?P ?A*)
proof −
  { from *assms* **have** *trans* [*unfolded transp-on-def*]: *transp-on A P transp-on B Q*
      by (*auto simp*: *wqo-on-def*)
    have *transp-on ?A ?P*
      unfolding *transp-on-def* **by** (*auto*, *insert trans*) (*blast*+) **}**
  moreover
  { from *assms* **and** *almost-full-on-Plus* **have** *almost-full-on ?P ?A* **by** (*auto simp*: *wqo-on-def*) **}**
  ultimately
  show *?thesis* **by** (*auto simp*: *wqo-on-def*)
qed

lemma *wqo-on-sum-UNIV* [*intro*]:
  wqo-on P UNIV $\Longrightarrow$ *wqo-on Q UNIV* $\Longrightarrow$ *wqo-on* (*sum-le P Q*) *UNIV*
  using *wqo-on-Plus* [*of P UNIV Q UNIV*] **by** *simp*

8.4 Dickson's Lemma

lemma *wqo-on-Sigma*:
  fixes *A1* :: *'a set* **and** *A2* :: *'b set*
  assumes *wqo-on P1 A1* **and** *wqo-on P2 A2*

shows *wqo-on* (*prod-le P1 P2*) (*A1* × *A2*) (**is** *wqo-on ?P ?A*)
proof −
{ **from** *assms* **have** *transp-on A1 P1* **and** *transp-on A2 P2* **by** (*auto simp*: *wqo-on-def*)
hence *transp-on ?A ?P* **unfolding** *transp-on-def prod-le-def* **by** *blast* }
moreover
{ **from** *assms* **and** *almost-full-on-Sigma* [*of P1 A1 P2 A2*]
have *almost-full-on ?P ?A* **by** (*auto simp*: *wqo-on-def*) }
ultimately
show *?thesis* **by** (*auto simp*: *wqo-on-def*)
qed

lemmas *dickson* = *wqo-on-Sigma*

lemma *wqo-on-prod-UNIV* [*intro*]:
wqo-on P UNIV ⟹ *wqo-on Q UNIV* ⟹ *wqo-on* (*prod-le P Q*) *UNIV*
using *wqo-on-Sigma* [*of P UNIV Q UNIV*] **by** *simp*

8.5 Higman's Lemma

lemma *transp-on-list-emb*:
assumes *transp-on A P*
shows *transp-on* (*lists A*) (*list-emb P*)
using *assms* **and** *list-emb-trans* [*of - - - P*]
unfolding *transp-on-def* **by** *blast*

lemma *wqo-on-lists*:
assumes *wqo-on P A* **shows** *wqo-on* (*list-emb P*) (*lists A*)
using *assms* **and** *almost-full-on-lists*
and *transp-on-list-emb* **by** (*auto simp*: *wqo-on-def*)

lemmas *higman* = *wqo-on-lists*

lemma *wqo-on-list-UNIV* [*intro*]:
wqo-on P UNIV ⟹ *wqo-on* (*list-emb P*) *UNIV*
using *wqo-on-lists* [*of P UNIV*] **by** *simp*

Every reflexive and transitive relation on a finite set is a wqo.

lemma *finite-wqo-on*:
assumes *finite A* **and** *refl*: *reflp-on A P* **and** *transp-on A P*
shows *wqo-on P A*
using *assms* **and** *finite-almost-full-on* **by** (*auto simp*: *wqo-on-def*)

lemma *finite-eq-wqo-on*:
assumes *finite A*
shows *wqo-on* (=) *A*
using *finite-wqo-on* [*OF assms*, *of* (=)]
by (*auto simp*: *reflp-on-def transp-on-def*)

lemma *wqo-on-lists-over-finite-sets*:
wqo-on (*list-emb* (=)) (*UNIV*::(*'a*::*finite*) *list set*)
using *wqo-on-lists* [*OF finite-eq-wqo-on* [*OF finite* [*of UNIV*::(*'a*::*finite*) *set*]]] **by** *simp*

lemma *wqo-on-map*:
fixes P **and** Q **and** h
defines $P' \equiv \lambda x\ y.\ P\ x\ y \wedge Q\ (h\ x)\ (h\ y)$
assumes *wqo-on* P A
and *wqo-on* Q B
and *subset*: h ‘ $A \subseteq B$
shows *wqo-on* P' A
proof
let $?Q = \lambda x\ y.\ Q\ (h\ x)\ (h\ y)$
from ‹*wqo-on* P A› **have** *transp-on* A P
by (*rule wqo-on-imp-transp-on*)
then show *transp-on* A P'
using ‹*wqo-on* Q B› **and** *subset*
unfolding *wqo-on-def transp-on-def* P'*-def* **by** *blast*

from ‹*wqo-on* P A› **have** *almost-full-on* P A
by (*rule wqo-on-imp-almost-full-on*)
from ‹*wqo-on* Q B› **have** *almost-full-on* Q B
by (*rule wqo-on-imp-almost-full-on*)

show *almost-full-on* P' A
proof
fix f
assume $*$: $\forall\, i$::*nat*. $f\ i \in A$
from *almost-full-on-imp-homogeneous-subseq* [*OF* ‹*almost-full-on* P A› *this*]
obtain g :: *nat* $\Rightarrow$ *nat*
where g: $\bigwedge i\ j.\ i < j \Longrightarrow g\ i < g\ j$
and $**$: $\forall\, i.\ f\ (g\ i) \in A \wedge P\ (f\ (g\ i))\ (f\ (g\ (Suc\ i)))$
using $*$ **by** *auto*
from *chain-transp-on-less* [*OF* $**$ ‹*transp-on* A P›]
have $**$: $\bigwedge i\ j.\ i < j \Longrightarrow P\ (f\ (g\ i))\ (f\ (g\ j))$ **.**
let $?g = \lambda i.\ h\ (f\ (g\ i))$
from $*$ **and** *subset* **have** B: $\bigwedge i.\ ?g\ i \in B$ **by** *auto*
with ‹*almost-full-on* Q B› [*unfolded almost-full-on-def good-def*, *THEN bspec*, *of ?g*]
obtain $i\ j$ :: *nat*
where $i < j$ **and** $Q\ (?g\ i)\ (?g\ j)$ **by** *blast*
with $**$ [*OF* ‹$i < j$›] **have** $P'\ (f\ (g\ i))\ (f\ (g\ j))$
by (*auto simp*: P'*-def*)
with g [*OF* ‹$i < j$›] **show** *good* P' f **by** (*auto simp*: *good-def*)
qed
qed

lemma *wqo-on-UNIV-nat*:

wqo-on $(\le)$ (*UNIV* :: *nat set*)
unfolding *wqo-on-def transp-on-def*
using *almost-full-on-UNIV-nat* **by** *simp*

end

9 Kruskal's Tree Theorem

theory *Kruskal*
imports *Well-Quasi-Orders*
begin

locale *kruskal-tree* =
fixes F :: ($'b \times nat$) *set*
and *mk* :: $'b \Rightarrow 'a$ *list* $\Rightarrow$ ($'a$::*size*)
and *root* :: $'a \Rightarrow 'b \times nat$
and *args* :: $'a \Rightarrow 'a$ *list*
and *trees* :: $'a$ *set*
assumes *size-arg*: $t \in$ *trees* $\Longrightarrow s \in$ *set* (*args* t) $\Longrightarrow$ *size* $s <$ *size* t
and *root-mk*: (f, *length ts*) $\in F \Longrightarrow$ *root* (*mk f ts*) = (f, *length ts*)
and *args-mk*: (f, *length ts*) $\in F \Longrightarrow$ *args* (*mk f ts*) = *ts*
and *mk-root-args*: $t \in$ *trees* $\Longrightarrow$ *mk* (*fst* (*root* t)) (*args* t) = t
and *trees-root*: $t \in$ *trees* $\Longrightarrow$ *root* $t \in F$
and *trees-arity*: $t \in$ *trees* $\Longrightarrow$ *length* (*args* t) = *snd* (*root* t)
and *trees-args*: $\bigwedge s.\ t \in$ *trees* $\Longrightarrow s \in$ *set* (*args* t) $\Longrightarrow s \in$ *trees*
begin

lemma *mk-inject* [*iff*]:
assumes (f, *length ss*) $\in F$ **and** (g, *length ts*) $\in F$
shows *mk f ss* = *mk g ts* $\longleftrightarrow f = g \wedge$ *ss* = *ts*
proof −
{ assume *mk f ss* = *mk g ts*
then have *root* (*mk f ss*) = *root* (*mk g ts*)
and *args* (*mk f ss*) = *args* (*mk g ts*) **by** *auto* **}**
show *?thesis*
using *root-mk* [*OF assms*(*1*)] **and** *root-mk* [*OF assms*(*2*)]
and *args-mk* [*OF assms*(*1*)] **and** *args-mk* [*OF assms*(*2*)] **by** *auto*
qed

inductive *emb* **for** P
where
arg: $[\![(f, m) \in F$; *length ts* = m; $\forall t{\in}$*set ts*. $t \in$ *trees*;
$t \in$ *set ts*; *emb P s t*$]\!] \Longrightarrow$ *emb P s* (*mk f ts*) |
list-emb: $[\![(f, m) \in F$; $(g, n) \in F$; *length ss* = m; *length ts* = n;
$\forall s \in$ *set ss*. $s \in$ *trees*; $\forall t \in$ *set ts*. $t \in$ *trees*;
P (f, m) (g, n); *list-emb* (*emb P*) *ss ts*$]\!] \Longrightarrow$ *emb P* (*mk f ss*) (*mk g ts*)
monos *list-emb-mono*

lemma *almost-full-on-trees*:

assumes *almost-full-on P F*
shows *almost-full-on* (*emb P*) *trees* (**is** *almost-full-on ?P ?A*)
proof (*rule ccontr*)
interpret *mbs ?A* **.**
assume ¬ *?thesis*
from *mbs′* [*OF this*] **obtain** *m*
where *bad*: *m* ∈ *BAD ?P*
and *min*: ∀ *g*. (*m*, *g*) ∈ *gseq* ⟶ *good ?P g* **..**
then have *trees*: ⋀*i*. *m i* ∈ *trees* **by** *auto*

define *r* **where** *r i* = *root* (*m i*) **for** *i*
define *a* **where** *a i* = *args* (*m i*) **for** *i*
define *S* **where** *S* = ⋃{*set* (*a i*) | *i*. *True*}

have *m*: ⋀*i*. *m i* = *mk* (*fst* (*r i*)) (*a i*)
by (*simp add*: *r-def a-def mk-root-args* [*OF trees*])
have *lists*: ∀ *i*. *a i* ∈ *lists S* **by** (*auto simp*: *a-def S-def*)
have *arity*: ⋀*i*. *length* (*a i*) = *snd* (*r i*)
using *trees-arity* [*OF trees*] **by** (*auto simp*: *r-def a-def*)
then have *sig*: ⋀*i*. (*fst* (*r i*), *length* (*a i*)) ∈ *F*
using *trees-root* [*OF trees*] **by** (*auto simp*: *a-def r-def*)
have *a-trees*: ⋀*i*. ∀ *t* ∈ *set* (*a i*). *t* ∈ *trees* **by** (*auto simp*: *a-def trees-args* [*OF trees*])

have *almost-full-on ?P S*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *s* :: *nat* ⇒ *′a*
where *S*: ⋀*i*. *s i* ∈ *S* **and** *bad-s*: *bad ?P s* **by** (*auto simp*: *almost-full-on-def*)

define *n* **where** *n* = (*LEAST n*. ∃ *k*. *s k* ∈ *set* (*a n*))
have ∃ *n*. ∃ *k*. *s k* ∈ *set* (*a n*) **using** *S* **by** (*force simp*: *S-def*)
from *LeastI-ex* [*OF this*] **obtain** *k*
where *sk*: *s k* ∈ *set* (*a n*) **by** (*auto simp*: *n-def*)
have *args*: ⋀*k*. ∃ *m* ≥ *n*. *s k* ∈ *set* (*a m*)
using *S* **by** (*auto simp*: *S-def*) (*metis Least-le n-def*)

define *m′* **where** *m′ i* = (*if i* < *n then m i else s* (*k* + (*i* − *n*))) **for** *i*

have *m′-less*: ⋀*i*. *i* < *n* ⟹ *m′ i* = *m i* **by** (*simp add*: *m′-def*)
have *m′-geq*: ⋀*i*. *i* ≥ *n* ⟹ *m′ i* = *s* (*k* + (*i* − *n*)) **by** (*simp add*: *m′-def*)

have *bad ?P m′*
proof
assume *good ?P m′*
then obtain *i j* **where** *i* < *j* **and** *emb*: *?P* (*m′ i*) (*m′ j*) **by** *auto*
{ assume *j* < *n*
with ‹*i* < *j*› **and** *emb* **have** *?P* (*m i*) (*m j*) **by** (*auto simp*: *m′-less*)
with ‹*i* < *j*› **and** *bad* **have** *False* **by** *blast* **}**

moreover
{ assume $n \leq i$
with ‹$i < j$› **and** *emb* **have** *?P* $(s\ (k + (i - n)))$ $(s\ (k + (j - n)))$
and $k + (i - n) < k + (j - n)$ **by** (*auto simp*: *m′-geq*)
with *bad-s* **have** *False* **by** *auto* **}**
moreover
{ assume $i < n$ **and** $n \leq j$
with ‹$i < j$› **and** *emb* **have** ∗: *?P* $(m\ i)$ $(s\ (k + (j - n)))$ **by** (*auto simp*: *m′-less m′-geq*)
with *args* **obtain** l **where** $l \geq n$ **and** ∗∗: $s\ (k + (j - n)) \in$ *set* $(a\ l)$ **by** *blast*
from *emb.arg* [*OF sig* [*of l*] - *a-trees* [*of l*] ∗∗ ∗]
have *?P* $(m\ i)$ $(m\ l)$ **by** (*simp add*: *m*)
moreover have $i < l$ **using** ‹$i < n$› **and** ‹$n \leq l$› **by** *auto*
ultimately have *False* **using** *bad* **by** *blast* **}**
ultimately show *False* **using** ‹$i < j$› **by** *arith*
qed
moreover have $(m, m') \in$ *gseq*
proof −
have $m \in$ *SEQ ?A* **using** *trees* **by** *auto*
moreover have $m' \in$ *SEQ ?A*
using *trees* **and** *S* **and** *trees-args* [*OF trees*] **by** (*auto simp*: *m′-def a-def S-def*)
moreover have $\forall\, i < n.\ m\ i = m'\ i$ **by** (*auto simp*: *m′-less*)
moreover have *size* $(m'\ n)$ < *size* $(m\ n)$
using *sk* **and** *size-arg* [*OF trees*, *unfolded m*]
by (*auto simp*: *m m′-geq root-mk* [*OF sig*] *args-mk* [*OF sig*])
ultimately show *?thesis* **by** (*auto simp*: *gseq-def*)
qed
ultimately show *False* **using** *min* **by** *blast*
qed
from *almost-full-on-lists* [*OF this*, *THEN almost-full-on-imp-homogeneous-subseq*, *OF lists*]
obtain φ :: *nat* ⇒ *nat*
where *less*: $\bigwedge i\ j.\ i < j \Longrightarrow \varphi\ i < \varphi\ j$
and *lemb*: $\bigwedge i\ j.\ i < j \Longrightarrow$ *list-emb ?P* $(a\ (\varphi\ i))$ $(a\ (\varphi\ j))$ **by** *blast*
have *roots*: $\bigwedge i.\ r\ (\varphi\ i) \in F$ **using** *trees* [*THEN trees-root*] **by** (*auto simp*: *r-def*)
then have $r \circ \varphi \in$ *SEQ F* **by** *auto*
with *assms* **have** *good P* $(r \circ \varphi)$ **by** (*auto simp*: *almost-full-on-def*)
then obtain *i j*
where $i < j$ **and** $P\ (r\ (\varphi\ i))\ (r\ (\varphi\ j))$ **by** *auto*
with *lemb* [*OF* ‹$i < j$›] **have** *?P* $(m\ (\varphi\ i))$ $(m\ (\varphi\ j))$
using *sig* **and** *arity* **and** *a-trees* **by** (*auto simp*: *m intro*!: *emb.list-emb*)
with *less* [*OF* ‹$i < j$›] **and** *bad* **show** *False* **by** *blast*
qed

inductive-cases
emb-mk2 [*consumes 1*, *case-names arg list-emb*]: *emb P s* (*mk g ts*)

inductive-cases
list-emb-Nil2-cases: *list-emb P xs* [] **and**
list-emb-Cons-cases: *list-emb P xs* (*y*#*ys*)

lemma *list-emb-trans-right*:
assumes *list-emb P xs ys* **and** *list-emb* ($\lambda y\ z.\ P\ y\ z \wedge (\forall x.\ P\ x\ y \longrightarrow P\ x\ z)$) *ys zs*
shows *list-emb P xs zs*
using *assms*(*2*, *1*) **by** (*induct arbitrary*: *xs*) (*auto elim*!: *list-emb-Nil2-cases list-emb-Cons-cases*)

lemma *emb-trans*:
assumes *trans*: $\bigwedge f\ g\ h.\ f \in F \Longrightarrow g \in F \Longrightarrow h \in F \Longrightarrow P\ f\ g \Longrightarrow P\ g\ h \Longrightarrow P\ f\ h$
assumes *emb P s t* **and** *emb P t u*
shows *emb P s u*
using *assms*(*3*, *2*)
proof (*induct arbitrary*: *s*)
case (*arg f m ts v*)
then show *?case* **by** (*auto intro*: *emb.arg*)
next
case (*list-emb f m g n ss ts*)
note *IH* = *this*
from ‹*emb P s* (*mk f ss*)›
show *?case*
proof (*cases rule*: *emb-mk2*)
case *arg*
then show *?thesis* **using** *IH* **by** (*auto elim*!: *list-emb-set intro*: *emb.arg*)
next
case *list-emb*
then show *?thesis* **using** *IH* **by** (*auto intro*: *emb.intros dest*: *trans list-emb-trans-right*)
qed
qed

lemma *transp-on-emb*:
assumes *transp-on F P*
shows *transp-on trees* (*emb P*)
using *assms* **and** *emb-trans* [*of P*] **unfolding** *transp-on-def* **by** *blast*

lemma *kruskal*:
assumes *wqo-on P F*
shows *wqo-on* (*emb P*) *trees*
using *almost-full-on-trees* [*of P*] **and** *assms* **by** (*metis transp-on-emb wqo-on-def*)

end

end
theory *Kruskal-Examples*
imports *Kruskal*

begin

datatype *'a tree* = *Node 'a 'a tree list*

fun *node*
where
  node (*Node f ts*) = (*f*, *length ts*)

fun *succs*
where
  succs (*Node f ts*) = *ts*

inductive-set *trees* **for** *A*
where
  $f \in A \Longrightarrow \forall\, t \in$ *set ts*. $t \in$ *trees* $A \Longrightarrow$ *Node f ts* $\in$ *trees A*

lemma [*simp*]:
  trees UNIV = *UNIV*
proof −
  { fix *t* :: *'a tree*
    have $t \in$ *trees UNIV*
      by (*induct t*) (*auto intro*: *trees.intros*) **}**
  then show *?thesis* **by** *auto*
qed

interpretation *kruskal-tree-tree*: *kruskal-tree A* × *UNIV Node node succs trees A*
for *A*
  apply (*unfold-locales*)
  apply *auto*
  apply (*case-tac* [!] *t rule*: *trees.cases*)
  apply *auto*
  by (*metis less-not-refl not-less-eq size-list-estimation*)

thm *kruskal-tree-tree.almost-full-on-trees*
thm *kruskal-tree-tree.kruskal*

definition *tree-emb A P* = *kruskal-tree-tree.emb A* (*prod-le P* (λ- -. *True*))

lemma *wqo-on-trees*:
  assumes *wqo-on P A*
  shows *wqo-on* (*tree-emb A P*) (*trees A*)
  using *wqo-on-Sigma* [*OF assms wqo-on-UNIV*, *THEN kruskal-tree-tree.kruskal*]
  by (*simp add*: *tree-emb-def*)

If the type *'a* is well-quasi-ordered by *P*, then trees of type *'a tree* are well-quasi-ordered by the homeomorphic embedding relation.

instantiation *tree* :: (*wqo*) *wqo*
begin
definition $s \le t \longleftrightarrow$ *tree-emb UNIV* ($\le$) *s t*

definition $(s :: 'a\ tree) < t \longleftrightarrow s \leq t \wedge \neg\ (t \leq s)$

instance
  by (*rule wqo.intro-of-class*)
    (*auto simp*: *less-eq-tree-def* [*abs-def*] *less-tree-def* [*abs-def*]
      intro: *wqo-on-trees* [*of - UNIV*, *simplified*])
end

datatype $('f, 'v)$ *term* = *Var* $'v$ | *Fun* $'f$ $('f, 'v)$ *term list*

fun *root*
where
  root (*Fun f ts*) = (*f*, *length ts*)

fun *args*
where
  args (*Fun f ts*) = *ts*

inductive-set *gterms* **for** *F*
where
  $(f, n) \in F \Longrightarrow$ *length ts* $= n \Longrightarrow \forall s \in$ *set ts*. $s \in$ *gterms* $F \Longrightarrow$ *Fun f ts* $\in$ *gterms* F

interpretation *kruskal-term*: *kruskal-tree F Fun root args gterms F* **for** *F*
  apply (*unfold-locales*)
  apply *auto*
  apply (*case-tac* [!] *t rule*: *gterms.cases*)
  apply *auto*
  by (*metis less-not-refl not-less-eq size-list-estimation*)

thm *kruskal-term.almost-full-on-trees*

inductive-set *terms*
where
  $\forall t \in$ *set ts*. $t \in$ *terms* $\Longrightarrow$ *Fun f ts* $\in$ *terms*

interpretation *kruskal-variadic*: *kruskal-tree UNIV Fun root args terms*
  apply (*unfold-locales*)
  apply *auto*
  apply (*case-tac* [!] *t rule*: *terms.cases*)
  apply *auto*
  by (*metis less-not-refl not-less-eq size-list-estimation*)

thm *kruskal-variadic.almost-full-on-trees*

datatype $'a$ *exp* = *V* $'a$ | *C nat* | *Plus* $'a$ *exp* $'a$ *exp*

datatype $'a$ *symb* = *v* $'a$ | *c nat* | *p*

fun *mk*
where
  $mk\ (v\ x)\ []\ =\ V\ x\ |$
  $mk\ (c\ n)\ []\ =\ C\ n\ |$
  $mk\ p\ [a,\ b]\ =\ Plus\ a\ b$

fun *rt*
where
  $rt\ (V\ x) = (v\ x,\ 0{::}nat)\ |$
  $rt\ (C\ n) = (c\ n,\ 0)\ |$
  $rt\ (Plus\ a\ b) = (p,\ 2)$

fun *ags*
where
  $ags\ (V\ x) = []\ |$
  $ags\ (C\ n) = []\ |$
  $ags\ (Plus\ a\ b) = [a,\ b]$

inductive-set *exps*
where
  $V\ x \in exps\ |$
  $C\ n \in exps\ |$
  $a \in exps \Longrightarrow b \in exps \Longrightarrow Plus\ a\ b \in exps$

lemma [*simp*]:
  assumes $length\ ts = 2$
  shows $rt\ (mk\ p\ ts) = (p,\ 2)$
  using *assms* **by** (*induct ts*) (*auto*, *case-tac ts*, *auto*)

lemma [*simp*]:
  assumes $length\ ts = 2$
  shows $ags\ (mk\ p\ ts) = ts$
  using *assms* **by** (*induct ts*) (*auto*, *case-tac ts*, *auto*)

interpretation *kruskal-exp*: *kruskal-tree*
  $\{(v\ x,\ 0)\ |\ x.\ True\} \cup \{(c\ n,\ 0)\ |\ n.\ True\} \cup \{(p,\ 2)\}$
  mk rt ags exps
apply (*unfold-locales*)
apply *auto*
apply (*case-tac* [!] *t rule*: *exps.cases*)
by *auto*

thm *kruskal-exp.almost-full-on-trees*

hide-const (**open**) *tree-emb V C Plus v c p*

end

10 Instances of Well-Quasi-Orders

theory *Wqo-Instances*
imports *Kruskal*
begin

10.1 The Option Type is Well-Quasi-Ordered

instantiation *option* :: (*wqo*) *wqo*
begin
definition $x \le y \longleftrightarrow$ *option-le* $(\le)$ x y
definition $(x :: 'a\ option) < y \longleftrightarrow x \le y \wedge \neg\ (y \le x)$

instance
  by (*rule wqo.intro-of-class*)
    (*auto simp*: *less-eq-option-def* [*abs-def*] *less-option-def* [*abs-def*])
end

10.2 The Sum Type is Well-Quasi-Ordered

instantiation *sum* :: (*wqo*, *wqo*) *wqo*
begin
definition $x \le y \longleftrightarrow$ *sum-le* $(\le)$ $(\le)$ x y
definition $(x :: 'a + 'b) < y \longleftrightarrow x \le y \wedge \neg\ (y \le x)$

instance
  by (*rule wqo.intro-of-class*)
    (*auto simp*: *less-eq-sum-def* [*abs-def*] *less-sum-def* [*abs-def*])
end

10.3 Pairs are Well-Quasi-Ordered

If types $'a$ and $'b$ are well-quasi-ordered by P and Q, then pairs of type $'a \times 'b$ are well-quasi-ordered by the pointwise combination of P and Q.

instantiation *prod* :: (*wqo*, *wqo*) *wqo*
begin
definition $p \le q \longleftrightarrow$ *prod-le* $(\le)$ $(\le)$ p q
definition $(p :: 'a \times 'b) < q \longleftrightarrow p \le q \wedge \neg\ (q \le p)$

instance
  by (*rule wqo.intro-of-class*)
    (*auto simp*: *less-eq-prod-def* [*abs-def*] *less-prod-def* [*abs-def*])
end

10.4 Lists are Well-Quasi-Ordered

If the type $'a$ is well-quasi-ordered by P, then lists of type $'a$ *list* are well-quasi-ordered by the homeomorphic embedding relation.

instantiation *list* :: (*wqo*) *wqo*
begin
definition $xs \le ys \longleftrightarrow$ *list-emb* $(\le)$ *xs ys*
definition $(xs :: 'a\ list) < ys \longleftrightarrow xs \le ys \wedge \neg\ (ys \le xs)$

instance
  by (*rule wqo.intro-of-class*)
    (*auto simp*: *less-eq-list-def* [*abs-def*] *less-list-def* [*abs-def*])
end

end

11 Multiset Extension of Orders (as Binary Predicates)

theory *Multiset-Extension*
imports
  Open-Induction.Restricted-Predicates
  HOL−Library.Multiset
begin

definition *multisets* :: $'a\ set \Rightarrow 'a\ multiset\ set$ **where**
  multisets $A = \{M.\ set\text{-}mset\ M \subseteq A\}$

lemma *in-multisets-iff*:
  $M \in multisets\ A \longleftrightarrow set\text{-}mset\ M \subseteq A$
  by (*simp add*: *multisets-def*)

lemma *empty-multisets* [*simp*]:
  $\{\#\} \in multisets\ F$
  by (*simp add*: *in-multisets-iff*)

lemma *multisets-union* [*simp*]:
  $M \in multisets\ A \Longrightarrow N \in multisets\ A \Longrightarrow M + N \in multisets\ A$
  by (*auto simp add*: *in-multisets-iff*)

definition *mulex1* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ **where**
  $mulex1\ P = (\lambda M\ N.\ (M,\ N) \in mult1\ \{(x,\ y).\ P\ x\ y\})$

lemma *mulex1-empty* [*iff*]:
  $mulex1\ P\ M\ \{\#\} \longleftrightarrow False$
  using *not-less-empty* [*of* $M\ \{(x,\ y).\ P\ x\ y\}$]
  by (*auto simp*: *mulex1-def*)

lemma *mulex1-add*: $mulex1\ P\ N\ (M0 + \{\#a\#\}) \Longrightarrow$
  $(\exists M.\ mulex1\ P\ M\ M0 \wedge N = M + \{\#a\#\}) \vee$
  $(\exists K.\ (\forall b.\ b \in\# K \longrightarrow P\ b\ a) \wedge N = M0 + K)$
  using *less-add* [*of* $N\ a\ M0\ \{(x,\ y).\ P\ x\ y\}$]

by (*auto simp*: *mulex1-def*)

lemma *mulex1-self-add-right* [*simp*]:
mulex1 P A (*add-mset a A*)
proof −
let *?R* = $\{(x, y).\ P\ x\ y\}$
thm *mult1-def*
have $A + \{\#a\#\} = A + \{\#a\#\}$ **by** *simp*
moreover have $A = A + \{\#\}$ **by** *simp*
moreover have $\forall b.\ b \in\# \{\#\} \longrightarrow (b, a) \in ?R$ **by** *simp*
ultimately have (*A*, *add-mset a A*) ∈ *mult1 ?R*
unfolding *mult1-def* **by** *blast*
then show *?thesis* **by** (*simp add*: *mulex1-def*)
qed

lemma *empty-mult1* [*simp*]:
$(\{\#\}, \{\#a\#\}) \in$ *mult1 R*
proof −
have $\{\#a\#\} = \{\#\} + \{\#a\#\}$ **by** *simp*
moreover have $\{\#\} = \{\#\} + \{\#\}$ **by** *simp*
moreover have $\forall b.\ b \in\# \{\#\} \longrightarrow (b, a) \in R$ **by** *simp*
ultimately show *?thesis* **unfolding** *mult1-def* **by** *force*
qed

lemma *empty-mulex1* [*simp*]:
mulex1 P $\{\#\}$ $\{\#a\#\}$
using *empty-mult1* [*of a* $\{(x, y).\ P\ x\ y\}$] **by** (*simp add*: *mulex1-def*)

definition *mulex-on* :: ($'a \Rightarrow 'a \Rightarrow bool$) $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ *bool* **where**
mulex-on P A = (*restrict-to* (*mulex1 P*) (*multisets A*))$^{++}$

abbreviation *mulex* :: ($'a \Rightarrow 'a \Rightarrow bool$) $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ $'a$ *multiset* $\Rightarrow$ *bool* **where**
mulex P ≡ *mulex-on P UNIV*

lemma *mulex-on-induct* [*consumes 1*, *case-names base step*, *induct pred*: *mulex-on*]:
assumes *mulex-on P A M N*
and $\bigwedge M\ N.$ ⟦*M* ∈ *multisets A*; *N* ∈ *multisets A*; *mulex1 P M N*⟧ ⟹ *Q M N*
and $\bigwedge L\ M\ N.$ ⟦*mulex-on P A L M*; *Q L M*; *N* ∈ *multisets A*; *mulex1 P M N*⟧ ⟹ *Q L N*
shows *Q M N*
using *assms* **unfolding** *mulex-on-def* **by** (*induct*) *blast+*

lemma *mulex-on-self-add-singleton-right* [*simp*]:
assumes *a* ∈ *A* **and** *M* ∈ *multisets A*
shows *mulex-on P A M* (*add-mset a M*)
proof −
have *mulex1 P M* $(M + \{\#a\#\})$ **by** *simp*

with *assms* **have** *restrict-to* (*mulex1 P*) (*multisets A*) *M* (*add-mset a M*)
by (*auto simp*: *multisets-def*)
then show *?thesis* **unfolding** *mulex-on-def* **by** *blast*
qed

lemma *singleton-multisets* [*iff*]:
{#*x*#} ∈ *multisets A* ⟷ *x* ∈ *A*
by (*auto simp*: *multisets-def*)

lemma *union-multisetsD*:
assumes *M* + *N* ∈ *multisets A*
shows *M* ∈ *multisets A* ∧ *N* ∈ *multisets A*
using *assms* **by** (*auto simp*: *multisets-def*)

lemma *mulex-on-multisetsD* [*dest*]:
assumes *mulex-on P F M N*
shows *M* ∈ *multisets F* **and** *N* ∈ *multisets F*
using *assms* **by** (*induct*) *auto*

lemma *union-multisets-iff* [*iff*]:
M + *N* ∈ *multisets A* ⟷ *M* ∈ *multisets A* ∧ *N* ∈ *multisets A*
by (*auto dest*: *union-multisetsD*)

lemma *add-mset-multisets-iff* [*iff*]:
add-mset a M ∈ *multisets A* ⟷ *a* ∈ *A* ∧ *M* ∈ *multisets A*
unfolding *add-mset-add-single*[*of a M*] *union-multisets-iff* **by** *auto*

lemma *mulex-on-trans*:
mulex-on P A L M ⟹ *mulex-on P A M N* ⟹ *mulex-on P A L N*
by (*auto simp*: *mulex-on-def*)

lemma *transp-on-mulex-on*:
transp-on B (*mulex-on P A*)
using *mulex-on-trans* [*of P A*] **by** (*auto simp*: *transp-on-def*)

lemma *mulex-on-add-right* [*simp*]:
assumes *mulex-on P A M N* **and** *a* ∈ *A*
shows *mulex-on P A M* (*add-mset a N*)
proof −
from *assms* **have** *a* ∈ *A* **and** *N* ∈ *multisets A* **by** *auto*
then have *mulex-on P A N* (*add-mset a N*) **by** *simp*
with ‹*mulex-on P A M N*› **show** *?thesis* **by** (*rule mulex-on-trans*)
qed

lemma *empty-mulex-on* [*simp*]:
assumes *M* ≠ {#} **and** *M* ∈ *multisets A*
shows *mulex-on P A* {#} *M*
using *assms*
proof (*induct M*)

```
  case (add a M)
  show ?case
  proof (cases M = {#})
    assume M = {#}
    with add show ?thesis by (auto simp: mulex-on-def)
  next
    assume M ≠ {#}
    with add show ?thesis by (auto intro: mulex-on-trans)
  qed
qed simp

lemma mulex-on-self-add-right [simp]:
  assumes M ∈ multisets A and K ∈ multisets A and K ≠ {#}
  shows mulex-on P A M (M + K)
using assms
proof (induct K)
  case empty
  then show ?case by (cases K = {#}) auto
next
  case (add a M)
  show ?case
  proof (cases M = {#})
    assume M = {#} with add show ?thesis by auto
  next
    assume M ≠ {#} with add show ?thesis
      by (auto dest: mulex-on-add-right simp add: ac-simps)
  qed
qed

lemma mult1-singleton [iff]:
  ({#x#}, {#y#}) ∈ mult1 R ⟷ (x, y) ∈ R
proof
  assume (x, y) ∈ R
  then have {#y#} = {#} + {#y#}
    and {#x#} = {#} + {#x#}
    and ∀ b. b ∈# {#x#} ⟶ (b, y) ∈ R by auto
  then show ({#x#}, {#y#}) ∈ mult1 R unfolding mult1-def by blast
next
  assume ({#x#}, {#y#}) ∈ mult1 R
  then obtain M0 K a
    where {#y#} = add-mset a M0
    and {#x#} = M0 + K
    and ∀ b. b ∈# K ⟶ (b, a) ∈ R
    unfolding mult1-def by blast
  then show (x, y) ∈ R by (auto simp: add-eq-conv-diff)
qed

lemma mulex1-singleton [iff]:
  mulex1 P {#x#} {#y#} ⟷ P x y
```

using *mult1-singleton* [*of* $x\ y$ $\{(x,\ y).\ P\ x\ y\}$] **by** (*simp add*: *mulex1-def*)

lemma *singleton-mulex-onI*:
$P\ x\ y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$ *mulex-on* $P\ A\ \{\#x\#\}\ \{\#y\#\}$
by (*auto simp*: *mulex-on-def*)

lemma *reflclp-mulex-on-add-right* [*simp*]:
assumes $(\textit{mulex-on}\ P\ A)^{==}\ M\ N$ **and** $M \in \textit{multisets}\ A$ **and** $a \in A$
shows *mulex-on* $P\ A\ M\ (N + \{\#a\#\})$
using *assms* **by** (*cases* $M = N$) *simp-all*

lemma *reflclp-mulex-on-add-right′* [*simp*]:
assumes $(\textit{mulex-on}\ P\ A)^{==}\ M\ N$ **and** $M \in \textit{multisets}\ A$ **and** $a \in A$
shows *mulex-on* $P\ A\ M\ (\{\#a\#\} + N)$
using *reflclp-mulex-on-add-right* [*OF assms*] **by** (*simp add*: *ac-simps*)

lemma *mulex-on-union-right* [*simp*]:
assumes *mulex-on* $P\ F\ A\ B$ **and** $K \in \textit{multisets}\ F$
shows *mulex-on* $P\ F\ A\ (K + B)$
using *assms*
proof (*induct* K)
case (*add* $a\ K$)
then have $a \in F$ **and** *mulex-on* $P\ F\ A\ (B + K)$ **by** (*auto simp*: *multisets-def ac-simps*)
then have *mulex-on* $P\ F\ A\ ((B + K) + \{\#a\#\})$ **by** *simp*
then show *?case* **by** (*simp add*: *ac-simps*)
qed *simp*

lemma *mulex-on-union-right′* [*simp*]:
assumes *mulex-on* $P\ F\ A\ B$ **and** $K \in \textit{multisets}\ F$
shows *mulex-on* $P\ F\ A\ (B + K)$
using *mulex-on-union-right* [*OF assms*] **by** (*simp add*: *ac-simps*)

Adapted from $wf\ ?r \Longrightarrow \forall M.\ M \in \textit{Wellfounded.acc}\ (\textit{mult1}\ ?r)$ in *HOL−Library.Multiset*.

lemma *accessible-on-mulex1-multisets*:
assumes *wf*: *wfp-on* $P\ A$
shows $\forall M{\in}\textit{multisets}\ A.$ *accessible-on* (*mulex1* P) (*multisets* A) M
proof
let *?P* = *mulex1* P
let *?A* = *multisets* A
let *?acc* = *accessible-on ?P ?A*
{
fix $M\ M0\ a$
assume *M0*: *?acc M0*
and $a \in A$
and $M0 \in\ ?A$
and *wf-hyp*: $\bigwedge b.\ [\![b \in A;\ P\ b\ a]\!] \Longrightarrow (\forall M.\ ?acc\ (M) \longrightarrow\ ?acc\ (M + \{\#b\#\}))$
and *acc-hyp*: $\forall M.\ M \in\ ?A \wedge\ ?P\ M\ M0 \longrightarrow\ ?acc\ (M + \{\#a\#\})$
then have *add-mset* $a\ M0 \in\ ?A$ **by** (*auto simp*: *multisets-def*)

```
  then have ?acc (add-mset a M0)
  proof (rule accessible-onI [of add-mset a M0])
    fix N
    assume N ∈ ?A
      and ?P N (add-mset a M0)
    then have ((∃M. M ∈ ?A ∧ ?P M M0 ∧ N = M + {#a#}) ∨
        (∃K. (∀b. b ∈# K ⟶ P b a) ∧ N = M0 + K))
      using mulex1-add [of P N M0 a] by (auto simp: multisets-def)
    then show ?acc (N)
    proof (elim exE disjE conjE)
      fix M assume M ∈ ?A and ?P M M0 and N: N = M + {#a#}
      from acc-hyp have M ∈ ?A ∧ ?P M M0 ⟶ ?acc (M + {#a#}) ..
      with ‹M ∈ ?A› and ‹?P M M0› have ?acc (M + {#a#}) by blast
      then show ?acc (N) by (simp only: N)
    next
      fix K
      assume N: N = M0 + K
      assume ∀b. b ∈# K ⟶ P b a
    moreover from N and ‹N ∈ ?A› have K ∈ ?A by (auto simp: multisets-def)
      ultimately have ?acc (M0 + K)
      proof (induct K)
        case empty
        from M0 show ?acc (M0 + {#}) by simp
      next
        case (add x K)
        from add.prems have x ∈ A and P x a by (auto simp: multisets-def)
        with wf-hyp have ∀M. ?acc M ⟶ ?acc (M + {#x#}) by blast
       moreover from add have ?acc (M0 + K) by (auto simp: multisets-def)
        ultimately show ?acc (M0 + (add-mset x K)) by simp
      qed
      then show ?acc N by (simp only: N)
    qed
  qed
} note tedious-reasoning = this

fix M
assume M ∈ ?A
then show ?acc M
proof (induct M)
  show ?acc {#}
  proof (rule accessible-onI)
    show {#} ∈ ?A by (auto simp: multisets-def)
  next
    fix b assume ?P b {#} then show ?acc b by simp
  qed
next
  case (add a M)
  then have ?acc M by (auto simp: multisets-def)
  from add have a ∈ A by (auto simp: multisets-def)
```

with *wf* **have** $\forall M.\ ?acc\ M \longrightarrow ?acc\ (add\text{-}mset\ a\ M)$
proof (*induct*)
case (*less a*)
then have *r*: $\bigwedge b.\ [\![b \in A;\ P\ b\ a]\!] \Longrightarrow (\forall M.\ ?acc\ M \longrightarrow ?acc\ (M + \{\#b\#\}))$
by *auto*
show $\forall M.\ ?acc\ M \longrightarrow ?acc\ (add\text{-}mset\ a\ M)$
proof (*intro allI impI*)
fix M'
assume $?acc\ M'$
moreover then have $M' \in ?A$ **by** (*blast dest*: *accessible-on-imp-mem*)
ultimately show $?acc\ (add\text{-}mset\ a\ M')$
by (*induct*) (*rule tedious-reasoning* [*OF* - ‹$a \in A$› - *r*], *auto*)
qed
qed
with ‹$?acc\ (M)$› **show** $?acc\ (add\text{-}mset\ a\ M)$ **by** *blast*
qed
qed

lemmas *wfp-on-mulex1-multisets* =
accessible-on-mulex1-multisets [*THEN accessible-on-imp-wfp-on*]

lemmas *irreflp-on-mulex1* =
wfp-on-mulex1-multisets [*THEN wfp-on-imp-irreflp-on*]

lemma *wfp-on-mulex-on-multisets*:
assumes *wfp-on P A*
shows *wfp-on* (*mulex-on P A*) (*multisets A*)
using *wfp-on-mulex1-multisets* [*OF assms*]
by (*simp only*: *mulex-on-def wfp-on-restrict-to-tranclp-wfp-on-conv*)

lemmas *irreflp-on-mulex-on* =
wfp-on-mulex-on-multisets [*THEN wfp-on-imp-irreflp-on*]

lemma *mulex1-union*:
$mulex1\ P\ M\ N \Longrightarrow mulex1\ P\ (K + M)\ (K + N)$
by (*auto simp*: *mulex1-def mult1-union*)

lemma *mulex-on-union*:
assumes *mulex-on P A M N* **and** $K \in multisets\ A$
shows $mulex\text{-}on\ P\ A\ (K + M)\ (K + N)$
using *assms*
proof (*induct*)
case (*base M N*)
then have $mulex1\ P\ (K + M)\ (K + N)$ **by** (*blast dest*: *mulex1-union*)
moreover from *base* **have** $(K + M) \in multisets\ A$
and $(K + N) \in multisets\ A$ **by** (*auto simp*: *multisets-def*)
ultimately have $restrict\text{-}to\ (mulex1\ P)\ (multisets\ A)\ (K + M)\ (K + N)$ **by**
auto
then show *?case* **by** (*auto simp*: *mulex-on-def*)

next
  case (*step L M N*)
  then have *mulex1 P* $(K + M)$ $(K + N)$ **by** (*blast dest*: *mulex1-union*)
  moreover from *step* **have** $(K + M) \in$ *multisets A* **and** $(K + N) \in$ *multisets A* **by** *blast+*
  ultimately have (*restrict-to* (*mulex1 P*) (*multisets A*))$^{++}$ $(K + M)$ $(K + N)$ **by** *auto*
  moreover have *mulex-on P A* $(K + L)$ $(K + M)$ **using** *step* **by** *blast*
  ultimately show *?case* **by** (*auto simp*: *mulex-on-def*)
qed

lemma *mulex-on-union′*:
  assumes *mulex-on P A M N* **and** $K \in$ *multisets A*
  shows *mulex-on P A* $(M + K)$ $(N + K)$
  using *mulex-on-union* [*OF assms*] **by** (*simp add*: *ac-simps*)

lemma *mulex-on-add-mset*:
  assumes *mulex-on P A M N* **and** $m \in A$
  shows *mulex-on P A* (*add-mset m M*) (*add-mset m N*)
  unfolding *add-mset-add-single*[*of m M*] *add-mset-add-single*[*of m N*]
  apply (*rule mulex-on-union′*)
  using *assms* **by** *auto*

lemma *union-mulex-on-mono*:
  mulex-on P F A C $\Longrightarrow$ *mulex-on P F B D* $\Longrightarrow$ *mulex-on P F* $(A + B)$ $(C + D)$
  by (*metis mulex-on-multisetsD mulex-on-trans mulex-on-union mulex-on-union′*)

lemma *mulex-on-add-mset′*:
  assumes *P m n* **and** $m \in A$ **and** $n \in A$ **and** $M \in$ *multisets A*
  shows *mulex-on P A* (*add-mset m M*) (*add-mset n M*)
  unfolding *add-mset-add-single*[*of m M*] *add-mset-add-single*[*of n M*]
  apply (*rule mulex-on-union*)
  using *assms* **by** (*auto simp*: *mulex-on-def*)

lemma *mulex-on-add-mset-mono*:
  assumes *P m n* **and** $m \in A$ **and** $n \in A$ **and** *mulex-on P A M N*
  shows *mulex-on P A* (*add-mset m M*) (*add-mset n N*)
  unfolding *add-mset-add-single*[*of m M*] *add-mset-add-single*[*of n N*]
  apply (*rule union-mulex-on-mono*)
  using *assms* **by** (*auto simp*: *mulex-on-def*)

lemma *union-mulex-on-mono1*:
  $A \in$ *multisets F* $\Longrightarrow$ (*mulex-on P F*)$^{==}$ *A C* $\Longrightarrow$ *mulex-on P F B D* $\Longrightarrow$
    mulex-on P F $(A + B)$ $(C + D)$
  by (*auto intro*: *union-mulex-on-mono mulex-on-union*)

lemma *union-mulex-on-mono2*:
  $B \in$ *multisets F* $\Longrightarrow$ *mulex-on P F A C* $\Longrightarrow$ (*mulex-on P F*)$^{==}$ *B D* $\Longrightarrow$
    mulex-on P F $(A + B)$ $(C + D)$

```
  by (auto intro: union-mulex-on-mono mulex-on-union′)

lemma mult1-mono:
  assumes ⋀x y. ⟦x ∈ A; y ∈ A; (x, y) ∈ R⟧ ⟹ (x, y) ∈ S
    and M ∈ multisets A
    and N ∈ multisets A
    and (M, N) ∈ mult1 R
  shows (M, N) ∈ mult1 S
  using assms unfolding mult1-def multisets-def
  by auto (metis (full-types) subsetD)

lemma mulex1-mono:
  assumes ⋀x y. ⟦x ∈ A; y ∈ A; P x y⟧ ⟹ Q x y
    and M ∈ multisets A
    and N ∈ multisets A
    and mulex1 P M N
  shows mulex1 Q M N
  using mult1-mono [of A {(x, y). P x y} {(x, y). Q x y} M N]
    and assms unfolding mulex1-def by blast

lemma mulex-on-mono:
  assumes *: ⋀x y. ⟦x ∈ A; y ∈ A; P x y⟧ ⟹ Q x y
    and mulex-on P A M N
  shows mulex-on Q A M N
proof −
  let ?rel = λP. (restrict-to (mulex1 P) (multisets A))
  from ‹mulex-on P A M N› have (?rel P)⁺⁺ M N by (simp add: mulex-on-def)
  then have (?rel Q)⁺⁺ M N
  proof (induct rule: tranclp.induct)
    case (r-into-trancl M N)
    then have M ∈ multisets A and N ∈ multisets A by auto
    from mulex1-mono [OF * this] and r-into-trancl
      show ?case by auto
  next
    case (trancl-into-trancl L M N)
    then have M ∈ multisets A and N ∈ multisets A by auto
    from mulex1-mono [OF * this] and trancl-into-trancl
      have ?rel Q M N by auto
    with ‹(?rel Q)⁺⁺ L M› show ?case by (rule tranclp.trancl-into-trancl)
  qed
  then show ?thesis by (simp add: mulex-on-def)
qed

lemma mult1-reflcl:
  assumes (M, N) ∈ mult1 R
  shows (M, N) ∈ mult1 (R⁼)
  using assms by (auto simp: mult1-def)

lemma mulex1-reflclp:
```

assumes *mulex1 P M N*
shows *mulex1* $(P^{==})$ *M N*
using *mulex1-mono* [*of UNIV P* $P^{==}$ *M N*, *OF - - - assms*]
by (*auto simp*: *multisets-def*)

lemma *mulex-on-reflclp*:
assumes *mulex-on P A M N*
shows *mulex-on* $(P^{==})$ *A M N*
using *mulex-on-mono* [*OF - assms*, *of* $P^{==}$] **by** *auto*

lemma *surj-on-multisets-mset*:
$\forall M \in multisets\ A.\ \exists xs \in lists\ A.\ M = mset\ xs$
proof
fix *M*
assume $M \in multisets\ A$
then show $\exists xs \in lists\ A.\ M = mset\ xs$
proof (*induct M*)
case *empty* **show** *?case* **by** *simp*
next
case (*add a M*)
then obtain *xs* **where** $xs \in lists\ A$ **and** $M = mset\ xs$ **by** *auto*
then have *add-mset a M* = *mset* (*a* # *xs*) **by** *simp*
moreover have *a* # *xs* ∈ *lists A* **using** ‹*xs* ∈ *lists A*› **and** *add* **by** *auto*
ultimately show *?case* **by** *blast*
qed
qed

lemma *image-mset-lists* [*simp*]:
mset ‘ *lists A* = *multisets A*
using *surj-on-multisets-mset* [*of A*]
by *auto* (*metis mem-Collect-eq multisets-def set-mset-mset subsetI*)

lemma *multisets-UNIV* [*simp*]: *multisets UNIV* = *UNIV*
by (*metis image-mset-lists lists-UNIV surj-mset*)

lemma *non-empty-multiset-induct* [*consumes 1*, *case-names singleton add*]:
assumes $M \neq \{\#\}$
and $\bigwedge x.\ P\ \{\#x\#\}$
and $\bigwedge x\ M.\ P\ M \Longrightarrow P\ (add\text{-}mset\ x\ M)$
shows *P M*
using *assms* **by** (*induct M*) *auto*

lemma *mulex-on-all-strict*:
assumes $X \neq \{\#\}$
assumes $X \in multisets\ A$ **and** $Y \in multisets\ A$
and $*$: $\forall y.\ y \in\#\ Y \longrightarrow (\exists x.\ x \in\#\ X \wedge P\ y\ x)$
shows *mulex-on P A Y X*
using *assms*
proof (*induction X arbitrary*: *Y rule*: *non-empty-multiset-induct*)

case (*singleton x*)
then have *mulex1 P Y* {#*x*#}
unfolding *mulex1-def mult1-def*
by *auto*
with *singleton* **show** *?case* **by** (*auto simp*: *mulex-on-def*)
next
case (*add x M*)
let $?Y = \{\# \ y \in\# \ Y. \ \exists x. \ x \in\# \ M \wedge P \ y \ x \ \#\}$
let $?Z = Y - ?Y$
have *Y*: $Y = ?Z + ?Y$ **by** (*subst multiset-eq-iff*) *auto*
from ‹$Y \in$ *multisets A*› **have** $?Y \in$ *multisets A* **by** (*metis multiset-partition union-multisets-iff*)
moreover have $\forall y. \ y \in\# \ ?Y \longrightarrow (\exists x. \ x \in\# \ M \wedge P \ y \ x)$ **by** *auto*
moreover have $M \in$ *multisets A* **using** *add* **by** *auto*
ultimately have *mulex-on P A ?Y M* **using** *add* **by** *blast*
moreover have *mulex-on P A ?Z* {#*x*#}
proof −
have $\{\#x\#\} = \{\#\} + \{\#x\#\}$ **by** *simp*
moreover have $?Z = \{\#\} + ?Z$ **by** *simp*
moreover have $\forall y. \ y \in\# \ ?Z \longrightarrow P \ y \ x$
using *add.prems* **by** (*auto simp add*: *in-diff-count split*: *if-splits*)
ultimately have *mulex1 P ?Z* {#*x*#} **unfolding** *mulex1-def mult1-def* **by** *blast*
moreover have $\{\#x\#\} \in$ *multisets A* **using** *add.prems* **by** *auto*
moreover have $?Z \in$ *multisets A*
using ‹$Y \in$ *multisets A*› **by** (*metis diff-union-cancelL multiset-partition union-multisetsD*)
ultimately show *?thesis* **by** (*auto simp*: *mulex-on-def*)
qed
ultimately have *mulex-on P A* $(?Y + ?Z)$ $(M + \{\#x\#\})$ **by** (*rule union-mulex-on-mono*)
then show *?case* **using** *Y* **by** (*simp add*: *ac-simps*)
qed

The following lemma shows that the textbook definition (e.g., "Term Rewriting and All That") is the same as the one used below.

lemma *diff-set-Ex-iff*:
$X \neq \{\#\} \wedge X \subseteq\# \ M \wedge N = (M - X) + Y \longleftrightarrow X \neq \{\#\} \wedge (\exists Z. \ M = Z + X \wedge N = Z + Y)$
by (*auto*) (*metis add-diff-cancel-left′ multiset-diff-union-assoc union-commute*)

Show that *mulex-on* is equivalent to the textbook definition of multiset-extension for transitive base orders.

lemma *mulex-on-alt-def*:
assumes *trans*: *transp-on A P*
shows *mulex-on P A M N* $\longleftrightarrow M \in$ *multisets A* $\wedge N \in$ *multisets A* $\wedge (\exists X \ Y \ Z.$
$X \neq \{\#\} \wedge N = Z + X \wedge M = Z + Y \wedge (\forall y. \ y \in\# \ Y \longrightarrow (\exists x. \ x \in\# \ X \wedge P \ y \ x)))$
(**is** $?P \ M \ N \longleftrightarrow ?Q \ M \ N$)

```
proof
  assume ?P M N then show ?Q M N
  proof (induct M N)
    case (base M N)
    then obtain a M0 K where N: N = M0 + {#a#}
      and M: M = M0 + K
      and *: ∀ b. b ∈# K ⟶ P b a
        and M ∈ multisets A and N ∈ multisets A by (auto simp: mulex1-def
mult1-def)
    moreover then have {#a#} ∈ multisets A and K ∈ multisets A by auto
    moreover have {#a#} ≠ {#} by auto
    moreover have N = M0 + {#a#} by fact
    moreover have M = M0 + K by fact
     moreover have ∀ y. y ∈# K ⟶ (∃ x. x ∈# {#a#} ∧ P y x) using * by
auto
    ultimately show ?case by blast
  next
    case (step L M N)
    then obtain X Y Z
      where L ∈ multisets A and M ∈ multisets A and N ∈ multisets A
      and X ∈ multisets A and Y ∈ multisets A
      and M: M = Z + X
      and L: L = Z + Y and X ≠ {#}
      and Y: ∀ y. y ∈# Y ⟶ (∃ x. x ∈# X ∧ P y x)
      and mulex1 P M N
      by blast
    from ‹mulex1 P M N› obtain a M0 K
      where N: N = add-mset a M0 and M': M = M0 + K
      and *: ∀ b. b ∈# K ⟶ P b a unfolding mulex1-def mult1-def by blast
    have L': L = (M − X) + Y by (simp add: L M)
    have K: ∀ y. y ∈# K ⟶ (∃ x. x ∈# {#a#} ∧ P y x) using * by auto
```

The remainder of the proof is adapted from the proof of Lemma 2.5.4. of the book "Term Rewriting and All That."

```
    let ?X = add-mset a (X − K)
    let ?Y = (K − X) + Y

    have L ∈ multisets A and N ∈ multisets A by fact+
    moreover have ?X ≠ {#} ∧ (∃ Z. N = Z + ?X ∧ L = Z + ?Y)
    proof −
      have ?X ≠ {#} by auto
      moreover have ?X ⊆# N
        using M N M' by (simp add: add.commute [of {#a#}])
          (metis Multiset.diff-subset-eq-self add.commute add-diff-cancel-right)
      moreover have L = (N − ?X) + ?Y
      proof (rule multiset-eqI)
        fix x :: 'a
        let ?c = λM. count M x
        let ?ic = λx. int (?c x)
```

from ‹$?X \subseteq\# N$› **have** $*$: $?c$ $\{\#a\#\}$ $+$ $?c$ $(X - K) \leq ?c\ N$
by (*auto simp add*: *subseteq-mset-def split*: *if-splits*)
from $*$ **have** $**$: $?c$ $(X - K) \leq ?c\ M0$ **unfolding** N **by** (*auto split*: *if-splits*)
have $?ic\ (N - ?X + ?Y) = int\ (?c\ N - ?c\ ?X) + ?ic\ ?Y$ **by** *simp*
also have $\ldots = int\ (?c\ N - (?c\ \{\#a\#\} + ?c\ (X - K))) + ?ic\ (K - X) + ?ic\ Y$ **by** *simp*
also have $\ldots = ?ic\ N - (?ic\ \{\#a\#\} + ?ic\ (X - K)) + ?ic\ (K - X) + ?ic\ Y$
using *of-nat-diff* $[OF\ *]$ **by** *simp*
also have $\ldots = (?ic\ N - ?ic\ \{\#a\#\}) - ?ic\ (X - K) + ?ic\ (K - X) + ?ic\ Y$ **by** *simp*
also have $\ldots = (?ic\ N - ?ic\ \{\#a\#\}) + (?ic\ (K - X) - ?ic\ (X - K)) + ?ic\ Y$ **by** *simp*
also have $\ldots = (?ic\ N - ?ic\ \{\#a\#\}) + (?ic\ K - ?ic\ X) + ?ic\ Y$ **by** *simp*
also have $\ldots = (?ic\ N - ?ic\ ?X) + ?ic\ ?Y$ **by** (*simp add*: N)
also have $\ldots = ?ic\ L$
unfolding L' M' N
using $**$ **by** (*simp add*: *algebra-simps*)
finally show $?c\ L = ?c\ (N - ?X + ?Y)$ **by** *simp*
qed
ultimately show *?thesis* **by** (*metis diff-set-Ex-iff*)
qed
moreover have $\forall y.\ y \in\# ?Y \longrightarrow (\exists x.\ x \in\# ?X \wedge P\ y\ x)$
proof (*intro allI impI*)
fix y **assume** $y \in\# ?Y$
then have $y \in\# K - X \vee y \in\# Y$ **by** *auto*
then show $\exists x.\ x \in\# ?X \wedge P\ y\ x$
proof
assume $y \in\# K - X$
then have $y \in\# K$ **by** (*rule in-diffD*)
with K **show** *?thesis* **by** *auto*
next
assume $y \in\# Y$
with Y **obtain** x **where** $x \in\# X$ **and** $P\ y\ x$ **by** *blast*
{ assume $x \in\# X - K$ **with** ‹$P\ y\ x$› **have** *?thesis* **by** *auto* **}**
moreover
{ assume $x \in\# K$ **with** $*$ **have** $P\ x\ a$ **by** *auto*
moreover have $y \in A$ **using** ‹$Y \in multisets\ A$› **and** ‹$y \in\# Y$› **by** (*auto simp*: *multisets-def*)
moreover have $a \in A$ **using** ‹$N \in multisets\ A$› **by** (*auto simp*: N)
moreover have $x \in A$ **using** ‹$M \in multisets\ A$› **and** ‹$x \in\# K$› **by** (*auto simp*: M' *multisets-def*)
ultimately have $P\ y\ a$ **using** ‹$P\ y\ x$› **and** *trans* **unfolding** *transp-on-def* **by** *blast*
then have *?thesis* **by** *force* **}**
moreover from ‹$x \in\# X$› **have** $x \in\# X - K \vee x \in\# K$
by (*auto simp add*: *in-diff-count not-in-iff*)
ultimately show *?thesis* **by** *auto*
qed

qed
ultimately show *?case* **by** *blast*
qed
next
assume *?Q M N*
then obtain *X Y Z* **where** $M \in$ *multisets A* **and** $N \in$ *multisets A*
and $X \neq \{\#\}$ **and** N: $N = Z + X$ **and** M: $M = Z + Y$
and $*$: $\forall y.\ y \in\# Y \longrightarrow (\exists x.\ x \in\# X \wedge P\ y\ x)$ **by** *blast*
with *mulex-on-all-strict* [*of X A Y*] **have** *mulex-on P A Y X* **by** *auto*
moreover from ‹$N \in$ *multisets A*› **have** $Z \in$ *multisets A* **by** (*auto simp*: *N*)
ultimately show *?P M N* **unfolding** *M N* **by** (*metis mulex-on-union*)
qed

end

12 Multiset Extension Preserves Well-Quasi-Orders

theory *Wqo-Multiset*
imports
Multiset-Extension
Well-Quasi-Orders
begin

lemma *list-emb-imp-reflclp-mulex-on*:
assumes $xs \in$ *lists A* **and** $ys \in$ *lists A*
and *list-emb P xs ys*
shows $(\textit{mulex-on}\ P\ A)^{==}$ (*mset xs*) (*mset ys*)
using *assms*(*3*, *1*, *2*)
proof (*induct*)
case (*list-emb-Nil ys*)
then show *?case*
by (*cases ys*) (*auto intro*!: *empty-mulex-on simp*: *multisets-def*)
next
case (*list-emb-Cons xs ys y*)
then show *?case* **by** (*auto intro*!: *mulex-on-self-add-singleton-right simp*: *multisets-def*)
next
case (*list-emb-Cons2 x y xs ys*)
then show *?case*
by (*force intro*: *union-mulex-on-mono mulex-on-add-mset*
mulex-on-add-mset′ mulex-on-add-mset-mono
simp: *multisets-def*)
qed

The (reflexive closure of the) multiset extension of an almost-full relation is almost-full.

lemma *almost-full-on-multisets*:
assumes *almost-full-on P A*
shows *almost-full-on* $(\textit{mulex-on}\ P\ A)^{==}$ (*multisets A*)

proof −
  let *?P* = (*mulex-on P A*)$^{==}$
  from *almost-full-on-hom* [*OF - almost-full-on-lists*, *of A P ?P mset*,
    OF list-emb-imp-reflclp-mulex-on, *simplified*]
    show *?thesis* **using** *assms* **by** *blast*
qed

lemma *wqo-on-multisets*:
  assumes *wqo-on P A*
  shows *wqo-on* (*mulex-on P A*)$^{==}$ (*multisets A*)
proof
  from *transp-on-mulex-on* [*of multisets A P A*]
    show *transp-on* (*multisets A*) (*mulex-on P A*)$^{==}$
    unfolding *transp-on-def* **by** *blast*
next
  from *almost-full-on-multisets* [*OF assms* [*THEN wqo-on-imp-almost-full-on*]]
    show *almost-full-on* (*mulex-on P A*)$^{==}$ (*multisets A*) **.**
qed

end

References

[1] C. S. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59(4):833–835, 1963. doi:10.1017/S0305004100003844.