# Well-Quasi-Orders 

Christian Sternagel*

March 11, 2024


#### 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

1 Infinite Sequences ..... 2
1.1 Lexicographic Order on Infinite Sequences ..... 3
2 Minimal elements of sets w.r.t. a well-founded and transi- tive relation ..... 4
3 Enumerations of Well-Ordered Sets in Increasing Order ..... 7
4 The Almost-Full Property ..... 8
4.1 Basic Definitions and Facts ..... 9
4.2 An equivalent inductive definition ..... 10
4.3 Special Case: Finite Sets ..... 16
4.4 Further Results ..... 17
5 Constructing Minimal Bad Sequences ..... 20

[^0]6 A Proof of Higman's Lemma via Open Induction ..... 23
6.1 Some facts about the suffix relation ..... 23
6.2 Lexicographic Order on Infinite Sequences ..... 24
7 Almost-Full Relations ..... 29
7.1 Adding a Bottom Element to a Set ..... 29
7.2 Adding a Bottom Element to an Almost-Full Set ..... 30
7.3 Disjoint Union of Almost-Full Sets ..... 30
7.4 Dickson's Lemma for Almost-Full Relations ..... 32
7.5 Higman's Lemma for Almost-Full Relations ..... 33
7.6 Natural Numbers ..... 34
8 Well-Quasi-Orders ..... 35
8.1 Basic Definitions ..... 35
8.2 Equivalent Definitions ..... 35
8.3 A Type Class for Well-Quasi-Orders ..... 37
8.4 Dickson's Lemma ..... 38
8.5 Higman's Lemma ..... 39
9 Kruskal's Tree Theorem ..... 41
10 Instances of Well-Quasi-Orders ..... 48
10.1 The Option Type is Well-Quasi-Ordered ..... 48
10.2 The Sum Type is Well-Quasi-Ordered ..... 48
10.3 Pairs are Well-Quasi-Ordered ..... 48
10.4 Lists are Well-Quasi-Ordered ..... 48
11 Multiset Extension of Orders (as Binary Predicates) ..... 49
12 Multiset Extension Preserves Well-Quasi-Orders ..... 62

## 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 $S E Q A=\left\{f:: n a t \Rightarrow{ }^{\prime} a . \forall i . f i \in A\right\}$
lemma $S E Q$-iff [iff]:
$f \in S E Q A \longleftrightarrow(\forall i . f i \in A)$
by (auto simp: SEQ-def)

The $i$-th "column" of a set $B$ of infinite sequences.
definition ith $B i=\{f i \mid f . f \in B\}$
lemma ithI [intro]:
$f \in B \Longrightarrow f i=x \Longrightarrow x \in$ ith $B i$
by (auto simp: ith-def)
lemma ithE [elim]:
$\llbracket x \in$ ith $B i ; \bigwedge f . \llbracket f \in B ; f i=x \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$
by (auto simp: ith-def)
lemma ith-conv:
$x \in i$ th $B i \longleftrightarrow(\exists f \in B . x=f i)$
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 eq-upto $::\left(n a t \Rightarrow{ }^{\prime} a\right)$ set $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow n a t \Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right)$ set where
eq-upto $B f i=\{g \in B . \forall j<i . f j=g j\}$
lemma eq-uptoI [intro]:
$\llbracket g \in B ; \bigwedge j . j<i \Longrightarrow f j=g j \rrbracket \Longrightarrow g \in$ eq-upto $B f i$
by (auto simp: eq-upto-def)
lemma eq-uptoE [elim]:
$\llbracket g \in$ eq-upto $B f i ; \llbracket g \in B ; \Lambda j . j<i \Longrightarrow f j=g j \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$
by (auto simp: eq-upto-def)
lemma eq-upto-Suc:
$\llbracket g \in$ eq-upto $B f i ; g i=f i \rrbracket \Longrightarrow g \in$ eq-upto $B f(S u c i)$
by (auto simp: eq-upto-def less-Suc-eq)
lemma eq-upto-0 [simp]:
eq-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 eq-upto $B f i=$ eq-upto $C g i$
using assms by (auto simp: eq-upto-def)

### 1.1 Lexicographic Order on Infinite Sequences

definition $L E X P f g \longleftrightarrow(\exists i::$ nat. $P(f i)(g i) \wedge(\forall j<i . f j=g j))$
abbreviation $L E X E Q P \equiv(L E X P)==$
lemma $L E X-i m p-n o t-L E X$ :
assumes $L E X P f g$

```
        and [dest]: \xyz.Pxy\LongrightarrowPyz\LongrightarrowPxz
        and [simp]: \x.\negP x x
    shows \negLEX P gf
proof -
    { fix i j :: nat
        assume P(fi) (gi) and \forallk<i.fk=gk
            and P(gj)(fj) and \forallk<j.gk=fk
        then have False by (cases i<j) (auto simp: not-less dest!: le-imp-less-or-eq)
}
    then show \negLEX P gf using <LEX Pfg` unfolding LEX-def by blast
qed
lemma LEX-cases:
    assumes LEX Pfg
    obtains (eq)f=g| (neq)k where }\foralli<k.fi=gi and P(fk)(gk
using assms by (auto simp: LEX-def)
lemma LEX-imp-less:
    assumes }\forallx\inA.\negPxx\mathrm{ and }f\inSEQA\veeg\inSEQ 
        and LEX Pfg}\mathrm{ and }\foralli<k.fi=gi and fk\not=g
    shows P(fk) (gk)
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 \(B=(S O M E x . x \in B \wedge(\forall y \in A . P y x \longrightarrow y \notin B))\)
lemma minimal:
    assumes \(x \in A\) and \(Q x\)
    shows \(\exists y \in A . P^{==} y x \wedge Q y \wedge(\forall z \in A . P z y \longrightarrow \neg Q z)\)
using \(w f\) and assms
proof (induction rule: wfp-on-induct)
    case (less \(x\) )
```

```
    then show ?case
    proof (cases }\forally\inA.Pyx\longrightarrow\negQy
        case True
        with less show ?thesis by blast
    next
    case False
    then obtain }y\mathrm{ where }y\inA\mathrm{ 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\subseteqA and B\not={}
    shows }\existsx.x\inB\wedge(\forally\inA.P y x\longrightarrowy\not\inB
using assms using minimal [of - \lambdax. x \in B] by auto
lemma min-elt-mem:
    assumes B\subseteqA and B\not={}
    shows min-elt B\inB
using someI-ex [OF min-elt-ex [OF assms]] by (auto simp: min-elt-def)
lemma min-elt-minimal:
    assumes *: B\subseteqA B\not={}
    assumes y}\inA\mathrm{ and }Py\mathrm{ (min-elt B)
    shows y}\not\in
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\subseteqSEQ A and C\not={}
    shows eq-upto C (lexmin C) i\not={}
proof (induct i)
    case 0
    show ?case using assms by auto
next
    let ?A = \lambdai. ith (eq-upto C (lexmin C) i) i
    case (Suc i)
    then have ?A i\not= {} by force
    moreover have eq-upto C (lexmin C) i\subseteqeq-upto C (lexmin C) 0 by auto
    ultimately have ?A i\subseteqA and ?A i\not={} using assms by (auto simp: ith-def)
    from min-elt-mem [OF this, folded lexmin]
        obtain f}\mathrm{ where f}\in\mathrm{ 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\subseteqSEQ A and C\not={}
    shows lexmin C ESEQ A
proof -
    {fix i
        let ? }X=ith (eq-upto C (lexmin C) i) i
        have ?X\subseteqA using assms by (auto simp: ith-def)
    moreover have ?X \not={} 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\subseteqSEQ A and C\not={}
    shows ith (eq-upto C (lexmin C) i) i\subseteqA
    and ith (eq-upto C (lexmin C) i) i\not={}
using eq-upto-lexmin-non-empty [OF assms, of i] and assms by (auto simp: ith-def)
lemma lexmin-minimal:
    C\subseteqSEQ A\LongrightarrowC\not={}\Longrightarrowy\inA\LongrightarrowPy(lexmin Ci)\Longrightarrowy\not\inith (eq-upto
C(lexmin C) i) i
using min-elt-minimal [OF non-empty-ith, folded lexmin].
lemma lexmin-mem:
    C\subseteqSEQ A\LongrightarrowC\not={}\Longrightarrowlexmin Ci\inith(eq-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 fi) (SEQ A)
    shows chain-on P (ith (eq-upto C fi) i) A
using assms
proof -
    { fix x y assume x fith(eq-upto Cfi) i and y fith(eq-upto C fi) i
        and }\negPxy\mathrm{ and }y\not=
    then obtain gh where *:g eq-upto C fih\ineq-upto Cfi
        and [simp]: x = giy=hi and eq: \forallj<i.gj=fj^hj=fj
        by (auto simp: ith-def eq-upto-def)
        with assms and }\langley\not=x\rangle\mathrm{ consider LEX P gh|LEX Phg by (force simp:
chain-on-def)
    then have P y x
    proof (cases)
        assume LEX P gh
        with eq and }\langley\not=x\rangle\mathrm{ 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 $\langle\neg P x y$ show $P y x$..
next
assume $L E X P h g$
with $e q$ and $\langle y \neq x\rangle$ show $P$ y 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<br>imports Main<br>begin

locale infinitely-many1 $=$
fixes $P$ :: ' $a$ :: wellorder $\Rightarrow$ bool
assumes infm: $\forall i . \exists j>i . P j$
begin
Enumerate the elements of a well-ordered infinite set in increasing order.
fun enum :: nat $\Rightarrow{ }^{\prime} a$ where
enum $0=(L E A S T$ n. $P n) \mid$
enum $($ Suc $i)=($ LEAST $n . n>$ enum $i \wedge P n)$
lemma enum-mono:
shows enum $i<$ enum (Suc $i$ )
using infm by (cases $i$, auto) (metis (lifting) LeastI)+
lemma enum-less:
$i<j \Longrightarrow$ 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 $\Rightarrow{ }^{\prime} a \Rightarrow$ bool and $N::{ }^{\prime} a$

```
    assumes infm: }\foralli\geqN.\existsj>i.Pi
```

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 }=>\mp@subsup{}{}{\prime}a\mathrm{ where
    enumchain 0 = N |
    enumchain (Suc n) = (LEAST m. m > enumchain n ^P(enumchain n)m)
lemma enumchain-mono:
    shows N\leqenumchain i}\wedge enumchain i<enumchain (Suc i
proof (induct i)
    case 0
    have enumchain 0\geqN by simp
    moreover then have \existsm>enumchain 0. P (enumchain 0) m}\mathrm{ using infm by
blast
    ultimately show ?case by auto (metis (lifting) LeastI)
next
    case (Suc i)
    then have N\leqenumchain (Suc i) by auto
    moreover then have \existsm>enumchain (Suc i). P (enumchain (Suc i))musing
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 \existsm>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 }\existsm>\mathrm{ 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 $y \longleftrightarrow x=0 \vee\left(\exists x^{\prime} . x=\right.$ Suc $\left.x^{\prime} \wedge x^{\prime} \leq y\right)$
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 $) \Rightarrow\left(\right.$ nat $\left.\Rightarrow^{\prime} a\right) \Rightarrow$ bool where

```
good Pf}\longleftrightarrow(\existsij.i<j^P(fi)(fj)
```

A sequence that is not good is called bad.
abbreviation $\operatorname{bad} P f \equiv \neg \operatorname{good} P f$
lemma goodI:
$\llbracket i<j ; P(f i)(f j) \rrbracket \Longrightarrow \operatorname{good} P f$
by (auto simp: good-def)
lemma goodE [elim]:
good $P f \Longrightarrow(\bigwedge i j . \llbracket i<j ; P(f i)(f j) \rrbracket \Longrightarrow Q) \Longrightarrow Q$
by (auto simp: good-def)
lemma badE [elim]:
$\operatorname{bad} P f \Longrightarrow((\bigwedge i j . i<j \Longrightarrow \neg P(f i)(f j)) \Longrightarrow Q) \Longrightarrow Q$
by (auto simp: good-def)
definition almost-full-on :: (' $a \Rightarrow^{\prime} a \Rightarrow$ bool $) \Rightarrow$ ' $a$ set $\Rightarrow$ bool where
almost-full-on $P A \longleftrightarrow(\forall f \in S E Q A$ good $P f)$

## lemma almost-full-onI [Pure.intro]:

$(\bigwedge f . \forall i . f i \in A \Longrightarrow$ good $P f) \Longrightarrow$ almost-full-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-full-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 of for $A$ where
now: $(\bigwedge x y, x \in A \Longrightarrow y \in A \Longrightarrow P x y) \Longrightarrow$ af $A P$
| later $:(\bigwedge x . x \in A \Longrightarrow a f A(\lambda y z . P y z \vee P x y)) \Longrightarrow a f A P$
lemma af-imp-almost-full-on:
assumes af $A P$
shows almost-full-on P A
proof
fix $f::$ nat $\Rightarrow{ }^{\prime} a$ 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 $[\operatorname{simp}]: g i=f($ 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 $\langle i<j\rangle$ 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\mathrm{ assume }x\in
    then show af A(\lambdayz.Q y z\veeQxy)
        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 (\lambdauv.\negPvu\vee\negPux)
    using assms
proof (induct)
    case (1 x)
    then have af A (\lambdauv.(\negPvu\vee\negPux)\vee\negPuy\vee\negPyx) if y\inA 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 (\lambdax y.\negP 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 => nat => 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\inA^\negaf A(\lambdayz.P yz\veePxy))
lemma not-af:
    \neg \text { af } A P \Longrightarrow ( \exists x y . x \in A \wedge ~ y \in A \wedge \neg P ~ P ~ y ) ~ \wedge ( \exists x \in A . \neg ~ a f ~ A ~ ( \lambda y z . P ~ y ~ z ~
\vee P x y))
    unfolding af.simps [of A P] by blast
fun }
    where
        F A P 0 = NOTAF A P
    |FAP(Suc i)=(let x=NOTAF A P in FA(\lambdayz.P yz\veePxy)i)
lemma almost-full-on-imp-af:
    assumes af: almost-full-on P A
    shows af A P
proof (rule ccontr)
    assume }\neg\mathrm{ af A P
    then have *:FAPn\inA^
        \negf A (\lambdayz.Pyz\vee (\existsi\leqn.P(FAPi) y)\vee (\existsj\leqn.\existsi.i<j^P(FAP
i) (FAPj))) for n
    proof (induct }n\mathrm{ arbitrary: P)
```

case 0
from $\neg \neg a f A P\rangle$ 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 $A P \in A \wedge \neg a f A(\lambda y z . P y z \vee P(N O T A F A P) y)$ unfolding NOTAF-def by (rule someI-ex)
with 0 show ?case by simp
next
case (Suc n)
from $\neg \neg a f A P\rangle$ have $\exists x . x \in A \wedge \neg a f A(\lambda y z . P y z \vee P x y)$ by (auto intro: af.intros)
then have NOTAF $A P \in A \wedge \neg a f A(\lambda y z . P y z \vee P(N O T A F A P) y)$ 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 $P=\lambda x$. $\neg a f A x]$ )
qed
then have $F A P \in S E Q A$ 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 ( $\lambda$ - -. 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 $\forall i:: n a t . f i \in A$

```
    shows \exists\varphi::nat => nat. \forallij.i<j\longrightarrow\varphii<\varphij\wedgeP(f(\varphi i)) (f(\varphij))
proof -
    define }X\mathrm{ where }X={{i,j}|ij::nat. i<j\wedgeP(fi)(fj)
    define }Y\mathrm{ where }Y=-
    define }h\mathrm{ where }h=(\lambdaZ\mathrm{ . if Z }\inX\mathrm{ then 0 else Suc 0)
    have [iff]: \x y. h {x,y}=0\longleftrightarrow \longleftrightarrow{x,y}\inX by (auto simp: h-def)
    have [iff]: \x y.h{x,y}=Suc 0\longleftrightarrow\longleftrightarrow < < , y}\inY by (auto simp:h-def Y-def)
    have }\forallx\inUNIV.\forally\inUNIV. x\not=y\longrightarrowh{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 *:}\forallx\inI.\forally\inI.x\not=y\longrightarrowh{x,y}=c\mathrm{ by blast
then interpret infinitely-many1 \lambdai.i}i\in
    by (unfold-locales) (simp add: infinite-nat-iff-unbounded)
    have c=0 \ c = 1 using <c<2` by arith
then show ?thesis
proof
    assume [simp]:c=0
    have }\forallij.i<j\longrightarrowP(f(enum i))(f(enum j))
    proof (intro allI impI)
        fix i j :: nat
        assume i<j
        from * and enum-P and enum-less [OF <i<j`] have {enum i, enum j}\in
X by auto
            with enum-less [OF<i<j`]
                show P (f (enum i)) (f (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 }\forallij.i<j\longrightarrow\negP(f(\mathrm{ enum i)) (f(enum j))
    proof (intro allI impI)
        fix ij :: nat
        assume i<j
        from * and enum-P and enum-less [OF <i<j\rangle] have {enum i, enum j} \in
Y by auto
        with enum-less [OF<i<j`]
            show }\negP(f(\mathrm{ enum i)) (f(enum j)) by (auto simp: Y-def X-def double-
ton-eq-iff)
    qed
    then have }\neg\mathrm{ good P (f○ enum) by auto
    moreover have }\foralli.f(\mathrm{ enum i) }\inA\mathrm{ 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 \negantichain-on PfA
proof
    assume *: antichain-on P f A
    then have }\foralli.fi\inA by sim
    with assms have good Pf by (auto simp: almost-full-on-def)
    then obtain ij where i<j and P(fi) (fj)
        unfolding good-def by auto
    moreover with * have incomparable P (fi) (fj) 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\subseteqB
    shows almost-full-on ( }\lambdaxy.Q(hx)(hy))A(\mathrm{ is almost-full-on ?P A)
proof
    fix f
    assume }\foralli::nat. fi\in
    then have }\i.h(fi)\inB\mathrm{ using }\langle\mp@subsup{h}{}{\prime}A\subseteqB\rangle\mathrm{ by auto
    with «almost-full-on Q B` [unfolded almost-full-on-def,THEN bspec, of h\circf]
        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=>'
    assumes hom: }\xy.\llbracketx\inA;y\inA;Pxy\rrbracket\LongrightarrowQ(hx)(hy
        and af: almost-full-on P A
    shows almost-full-on Q (h`A)
proof
    fix f :: nat # 'b
    assume }\foralli.fi\inh'
    then have \foralli. \existsx. x\inA\wedgefi=hx by (auto simp: image-def)
    from choice [OF this] obtain g
        where *: \foralli.gi\inA\wedgefi=h(gi) by blast
    show good Q f
    proof (rule ccontr)
        assume bad: bad Q f
        { fix ij :: nat
            assume i<j
            from bad have }\negQ(fi)(fj) using <i<j> by (auto simp: good-def
            with hom have \negP (gi) (gj) 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: $\backslash x y . \llbracket x \in A ; y \in A \rrbracket \Longrightarrow P x y=Q(h x)(h y)$ bij-betwh $A B$ and af: almost-full-on $Q B$
shows almost-full-on PA
proof
fix $f::$ nat $\Rightarrow{ }^{\prime} a$
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 Pf
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 $f:: n a t \Rightarrow{ }^{\prime} a$ 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 «totalp-on $A P$ have $\forall i j . i<j \longrightarrow P(f j)(f i)$
unfolding totalp-on-def by blast
then have $\bigwedge i . P(f(S u c i))(f i)$ by auto
with $\langle w f p$-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) (fi) (f (Suc i))
unfolding assms by auto
ultimately show False
unfolding good-def by auto
qed

```
lemma ne-lists:
    assumes }xs\not=[] and xs\in lists 
    shows hd xs\inA 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 \{x y xs ys. \llbracketP x y; list-emb P xs ys; Q xs ys\rrbracket\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 \foralli<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< 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 \(f\) :: nat \(\Rightarrow{ }^{\prime} a\)
    assume \(*: \forall i . f i \in A\)
    let \(? I=U N I V::\) 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 \(\langle k<l\rangle\) 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: $\Lambda x y . P x y \Longrightarrow Q x y$
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 $\exists f . \forall i . f i \in A \wedge \operatorname{strict} Q(f(S u c i))(f i)$
then obtain $f$ where $*: \forall i$. $f i \in A \wedge\left((\operatorname{strict} Q)^{-1-1}\right)(f i)(f($ 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 $A\left((\text { strict } Q)^{-1-1}\right)$ by simp
assume $\neg$ wfp-on $($ strict $Q) A$
then obtain $f:: n a t \Rightarrow{ }^{\prime} a$ where $A: \bigwedge i . f i \in A$
and $\forall i$. strict $Q(f(S u c i))(f i)$ unfolding wfp-on-def by blast+
then have $\forall i . f i \in A \wedge\left((\text { strict } Q)^{-1-1}\right)(f i)(f($ 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 \(\bigwedge i . P(f(g(S u c i)))(f(g i))\) 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 \(\varphi\) where \([\operatorname{simp}]: \varphi=(\lambda i .(h \leadsto\) Suc i) 0)
    have \(\bigwedge i . \varphi i<\varphi\) (Suc \(i)\)
        using \(\langle\forall k . h k>k\rangle\) 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 \(\Rightarrow{ }^{\prime} a\)
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=( }\lambdaxy.\existsi.x=f(Suc i)^y=fi
    define P' where P' = restrict-to P A
    define Q where [simp]: Q = (sup P'D)**
    have **: \ij.(DOO P**)++ (fi) (fj)\Longrightarrowi>j
    proof -
    fix ij
    assume (DOO P***)++}(fi)(fj
    then show i>j
        apply (induct fi fj 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 }\forallx\inA.\forally\inA.Pxy\longrightarrowQxy\mathrm{ 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 }\foralli.fi\inA\wedge\operatorname{strict Q (f (Suc i)) (fi)
    proof
        fix i
        have }\negQ(fi)(f(Suc i)
        proof
            assume Q (fi)(f(Suc i))
            then have (sup P'D)** (fi) (f (Suc i)) by auto
            moreover have (sup P' D)** = sup (P**) (P** OO (DOO P P**)}\mp@subsup{)}{}{++}
            proof -
                have }\bigwedgeAB.(A\cupB\mp@subsup{)}{}{*}=\mp@subsup{A}{}{*}\cup\mp@subsup{A}{}{*}O(BO\mp@subsup{A}{}{*}\mp@subsup{)}{}{+}\mathrm{ by regexp
            from this [to-pred] show ?thesis by blast
            qed
            ultimately have sup ( }\mp@subsup{P}{}{***})(\mp@subsup{P}{}{***}OO(DOO PO**)++)(fi)(f (Suc i)
by simp
            then have ( (P** OO (DOO P***)++})(fi)(f(Suc i)) by aut
            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 fi\inA^ strict Q (f (Suc i))(fi) by auto
    qed
    ultimately show False unfolding wfp-on-def by blast
    qed
qed
end
```


## 5 Constructing Minimal Bad Sequences

theory Minimal-Bad-Sequences<br>imports<br>Almost-Full<br>Minimal-Elements<br>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 \(m b s=\)
    fixes \(A::\left({ }^{\prime} a::\right.\) 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\inA\mathrm{ and }P
    shows \existsy\inA. size }y\leq\mathrm{ size }x\wedgePy\wedge(\forallz\inA\mathrm{ . size z< size }y\longrightarrow\negPz
using assms
proof (induction x taking: size rule: measure-induct)
    case (1 x)
    then show ?case
    proof (cases }\forally\inA\mathrm{ . size }y<\mathrm{ size }x\longrightarrow\negPy
        case True
        with 1 show ?thesis by blast
    next
        case False
        then obtain y where y\inA 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\inA\Longrightarrow size }x<\mathrm{ size }y\Longrightarrowx=y\Longrightarrow\mathrm{ False
    by simp
```

The set of all bad sequences over $A$.
definition $B A D P=\{f \in S E Q$ A. bad $P f\}$
lemma BAD-iff [iff]:
$f \in B A D P \longleftrightarrow(\forall i . f i \in A) \wedge$ bad $P f$
by (auto simp: BAD-def)

A partial order on infinite bad sequences.
definition geseq $::\left(\left(n a t \Rightarrow{ }^{\prime} a\right) \times\left(n a t \Rightarrow{ }^{\prime} a\right)\right)$ set
where

$$
\text { geseq }=
$$

```
    \(\{(f, g) . f \in S E Q A \wedge g \in S E Q A \wedge(f=g \vee(\exists i\). size \((g i)<\operatorname{size}(f i) \wedge(\forall j\)
\(<i . f j=g j)))\}\)
```

The strict part of the above order.

```
definition \(g s e q::\left(\left(n a t \Rightarrow{ }^{\prime} a\right) \times\left(n a t \Rightarrow{ }^{\prime} a\right)\right)\) set where
    gseq \(=\{(f, g) . f \in S E Q A \wedge g \in S E Q A \wedge(\exists\) i. size \((g i)<\) size \((f i) \wedge(\forall j<\)
i. \(f j=g j))\}\)
lemma geseq-iff:
    \((f, g) \in\) geseq \(\longleftrightarrow\)
        \(f \in S E Q A \wedge g \in S E Q A \wedge(f=g \vee(\exists i\). size \((g i)<\operatorname{size}(f i) \wedge(\forall j<i . f j\)
\(=g j))\) )
    by (auto simp: geseq-def)
lemma gseq-iff:
    \((f, g) \in g s e q \longleftrightarrow f \in S E Q A \wedge g \in S E Q A \wedge(\exists i\). size \((g i)<\operatorname{size}(f i) \wedge(\forall j\)
\(<i . f j=g j)\) )
    by (auto simp: gseq-def)
lemma geseqE:
    assumes \((f, g) \in\) geseq
        and \(\llbracket \forall i . f i \in A ; \forall i . g i \in A ; f=g \rrbracket \Longrightarrow Q\)
    and \(\bigwedge i . \llbracket \forall i . f i \in A ; \forall i . g i \in A ;\) size \((g i)<\operatorname{size}(f i) ; \forall j<i . f j=g j \rrbracket \Longrightarrow\)
\(Q\)
    shows \(Q\)
    using assms by (auto simp: geseq-iff)
lemma gseqE:
    assumes \((f, g) \in g s e q\)
        and \(\bigwedge i . \llbracket \forall i . f i \in A ; \forall i . g i \in A ;\) size \((g i)<\operatorname{size}(f i) ; \forall j<i . f j=g j \rrbracket \Longrightarrow\)
\(Q\)
    shows \(Q\)
    using assms by (auto simp: gseq-iff)
sublocale min-elt-size?: minimal-element measure-on size UNIV A
rewrites measure-on size \(U N I V \equiv \lambda x y\). 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 \(P::^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\) bool
begin
```

A lower bound to all sequences in a set of sequences $B$.
abbreviation $l b \equiv \operatorname{lexmin}(B A D P)$

## lemma eq-upto-BAD-mem:

assumes $f \in$ eq-upto $(B A D P) g i$
shows $f j \in A$
using assms by (auto)
Assume that there is some infinite bad sequence $h$.

```
context
    fixes }h:: nat => ' 'a
    assumes BAD-ex: h\inBAD P
begin
```

When there is a bad sequence, then filtering $B A D P$ w.r.t. positions in $l b$ never yields an empty set of sequences.
lemma eq-upto-BAD-non-empty:
eq-upto $(B A D P) l b i \neq\{ \}$
using eq-upto-lexmin-non-empty $[$ of $B A D P$ ] and $B A D$-ex by auto
lemma non-empty-ith:
shows ith (eq-upto $(B A D P) l b i) i \subseteq A$
and ith (eq-upto $(B A D P) l b i) i \neq\{ \}$
using eq-upto-BAD-non-empty [of $i]$ by auto

## lemmas

$l b$-minimal $=$ min-elt-minimal $[$ OF non-empty-ith, folded lexmin $]$ and
$l b-m e m=$ min-elt-mem $[$ OF non-empty-ith, folded lexmin $]$
$l b$ is a infinite bad sequence.
lemma $l b-B A D$ :
$l b \in B A D P$
proof -
have $*: ~ \bigwedge j . l b j \in$ ith (eq-upto $(B A D P) l b j) j$ 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 $l b$
then obtain $i j$ where $i<j$ and $P(l b i)(l b j)$ by (auto simp: good-def) from $*$ have $l b j \in$ ith (eq-upto $(B A D P) l b j) j$ by (auto)
then obtain $g$ where $g \in$ eq-upto $(B A D P) l b j$ and $g j=l b j$ by force
then have $\forall k \leq j . g k=l b k$ by (auto simp: order-le-less)
with $\langle i<j\rangle$ and $\langle P(l b i)(l b j)\rangle$ have $P(g i)(g j)$ by auto
with $\langle i\langle j\rangle$ have good $P g$ by (auto simp: good-def)
with $\langle g \in$ eq-upto $(B A D P) l b j\rangle$ have False by auto \}
ultimately show ?thesis by blast
qed
There is no infinite bad sequence that is strictly smaller than $l b$.

```
lemma lb-lower-bound:
    \forallg. (lb,g) G gseq \longrightarrowg\not\inBADP
proof (intro allI impI)
```

fix $g$
assume $(l b, g) \in$ gseq
then obtain $i$ where $g i \in A$ and size $(g i)<$ size $(l b i)$
and $\forall j<i . l b j=g j$ by (auto simp: gseq-iff)
moreover with lb-minimal
have $g i \notin i$ th (eq-upto $(B A D P) l b i) i$ by auto
ultimately show $g \notin B A D P$ by blast
qed
If there is at least one bad sequence, then there is also a minimal one.
lemma lower-bound-ex:
$\exists f \in B A D P . \forall g .(f, g) \in g s e q \longrightarrow g \notin B A D P$
using $l b-B A D$ and $l b$-lower-bound by blast
lemma gseq-conv:
$(f, g) \in g s e q \longleftrightarrow f \neq g \wedge(f, g) \in$ geseq
by (auto simp: gseq-def geseq-def dest: less-not-eq)
There is a minimal bad sequence.

```
lemma mbs:
\(\exists f \in B A D P . \forall g .(f, g) \in g s e q \longrightarrow \operatorname{good} P g\)
using lower-bound-ex by (auto simp: gseq-conv geseq-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 fg assume SEQ: f\inSEQ Ag\inSEQ A and LEXPfg and LEXP gf
    then obtain ij where P(fi)(gi) and P(gj) (fj)
        and}\forallk<i.fk=gk\mathrm{ and }\forallk<j.gk=fk by (auto simp:LEX-def
    then have P(f(min ij))(f(min ij))
        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\mathrm{ by (auto simp: irreflp-on-def)
qed
lemma LEX-trans:
    assumes transp-on A P and f\inSEQ A and g\inSEQ A and h\inSEQ A
        and LEXPfg}\mathrm{ and LEXPgh
    shows LEX Pfh
using assms by (auto simp: LEX-def transp-on-def) (metis less-trans linorder-neqE-nat)
lemma qo-on-LEXEQ:
    transp-on A P\Longrightarrowqo-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\not={}
    shows glb (LEX P) C (lexmin C)
proof
    have C\subseteqSEQ A using assms by (auto simp: chain-on-def)
    then have lexmin C \inSEQ A using {C\not={}` by (intro lexmin-SEQ-mem)
    note * = <C\subseteqSEQ A>\langleC\not={}\rangle
    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}\mathrm{ assume f}\in
        then show LEXEQ P (lexmin C) f
        proof (cases f = lexmin C)
            define i where i=(LEAST i.fi\not= lexmin C i)
            case False
            then have neq: \existsi.fi\not=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: $f i \neq l$ lexmin $C i$ and eq: $\forall j<i . f j=l e x m i n ~ C j$ by auto
then have $* *: f \in$ eq-upto $C(l e x m i n C)$ if $i \in i t h($ eq-upto $C(l e x m i n ~ C) i)$
using $\langle f \in C\rangle$ by force +
moreover from $* *$ have $\neg P(f i)($ lexmin $C i)$
using lexmin-minimal $[O F *$, of $f i j$ and $\langle f \in C\rangle$ and $\langle C \subseteq S E Q A\rangle$ by blast
moreover obtain $g$ where $g \in$ eq-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 $\langle C \subseteq S E Q A\rangle$ and $\operatorname{assms}(1)$ and lex $[o f g f i]$ and lex $[o f f$
$g i]$
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 $l b: l b(L E X P) C f$
then show LEXEQ Pf(lexmin C)
proof (cases $f=$ lexmin $C$ )
define $i$ where $i=($ LEAST $i . f i \neq$ lexmin $C i)$
case False
then have neq: $\exists i . f i \neq l e x m i n ~ C i$ 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: $f i \neq l e x m i n ~ C i$ and $e q: \forall j<i . f j=l e x m i n ~ C j$ by auto
obtain $h$ where $h \in$ eq-upto $C$ (lexmin C) (Suc i) and $h \in C$
using eq-upto-lexmin-non-empty $[O F *]$ by (auto simp: eq-upto-def)
then have $[\operatorname{simp}]: ~ \bigwedge j . j<S u c i \Longrightarrow h j=$ lexmin $C j$ by auto
with $l b$ and $\langle h \in C\rangle$ have LEX $P f h$ using neq by (auto simp: lb-def)
then have $P(f i)(h i)$
using $n e q$ and $e q$ and $\langle C \subseteq S E Q A\rangle$ and $\langle h \in C\rangle$ by (intro lex) auto
with eq show ?thesis by (auto simp: LEX-def)
qed $\operatorname{simp}$
qed
lemma dc-on-LEXEQ:
dc-on (LEXEQ P) (SEQ A)
proof
fix $C$ assume chain-on $(L E X E Q P) C(S E Q A)$ and $C \neq\{ \}$
then have chain: chain-on (LEX P) C (SEQ A) by (auto simp: chain-on-def)
then have $C \subseteq S E Q A$ by (auto simp: chain-on-def)
then have lexmin $C \in S E Q A$ using $\langle C \neq\{ \}\rangle$ by (intro lexmin-SEQ-mem)
have glb $(L E X P) C$ (lexmin $C)$ by (rule glb-LEX-lexmin $[$ OF chain $\langle C \neq\{ \}\rangle]$ )
then have $g l b(L E X E Q P) C$ (lexmin $C$ ) by (auto simp: glb-def lb-def)
with «lexmin $C \in S E Q$ A〉 show $\exists f \in S E Q$ A. glb (LEXEQ P) Cf 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:
    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 \(f: f \in S E Q A\)
    show good \(Q f=(\exists n . \forall g \in S E Q A .(\forall i<n . g i=f i) \longrightarrow\) good \(Q g)\)
    proof
        assume good \(Q f\)
        then obtain \(i\) and \(j\) where \(*: i<j Q(f i)(f j)\) by auto
        have \(\forall g \in S E Q A\). \((\forall i<S u c j . g i=f i) \longrightarrow\) good \(Q g\)
        proof (intro balli impI)
            fix \(g\) assume \(g \in S E Q A\) and \(\forall i<S u c j . g i=f i\)
            then show good \(Q\) g using * by (force simp: good-def)
        qed
        then show \(\exists n . \forall g \in S E Q A .(\forall i<n . g i=f i) \longrightarrow \operatorname{good} Q g .\).
    next
        assume \(\exists n . \forall g \in S E Q A\). \((\forall i<n . g i=f i) \longrightarrow \operatorname{good} Q g\)
        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 \(Q(S E Q A)\)
    shows open-on \((L E X E Q P) Q(S E Q A)\)
proof
    fix \(C\) assume chain: chain-on \((L E X E Q P) C(S E Q A)\) and ne: \(C \neq\{ \}\)
        and \(\exists g \in S E Q\) A. glb (LEXEQ P) \(C g \wedge Q g\)
    then obtain \(g\) where \(g: g \in S E Q A\) and \(g l b(L E X E Q P) C g\)
        and \(Q: Q g\) by blast
    then have \(g l b: g l b(L E X P) C g\) by (auto simp: glb-def lb-def)
    from chain have chain-on (LEX P) C (SEQ A) and \(C: C \subseteq S E Q A\) by (auto
simp: chain-on-def)
    note \(*=g l b-L E X-l e x m i n ~[O F ~ t h i s(1) n e] ~\)
    have lexmin \(C \in S E Q A\) 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 pol]
    have [simp]: lexmin C=g by auto
    from assms [THEN pt-open-onD,OF Q g]
    obtain n :: nat where **: \h. h \inSEQ A\Longrightarrow(\foralli<n.h i=gi)\longrightarrowQ h by
blast
    from eq-upto-lexmin-non-empty [OF C ne, of n]
    obtain f}\mathrm{ where f}\in\mathrm{ eq-upto C gn by auto
    then have f\inC and Qf using ** [of f] and C by force+
    then show }\existsf\inC.Qf\mathrm{ 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}\mathrm{ and }P\equiv(\lambdaxy.False) and Q\equiv(\lambdaf.\foralli.fi=b
    assumes [simp]: a\not=b
    shows minimal-element P A and open-on (LEXEQ P) Q (SEQ A)
        and }\neg\mathrm{ 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 \neg pt-open-on Q (SEQ A)
    proof
        define f:: nat => ' }a\mathrm{ where }f\equiv(\lambdax.b
        have f}\inSEQ A by (auto simp: A-def f-def
        moreover assume pt-open-on Q (SEQ A)
        ultimately have Q f \longleftrightarrow (\existsn. (\forallg\inSEQ A. (\foralli<n.gi=fi)\longrightarrowQg))
            unfolding pt-open-on-def by blast
    moreover have Qf by (auto simp: Q-def f-def)
    moreover have }\existsg\inSEQ A.(\foralli<n.gi=fi)\wedge\negQg\mathrm{ 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 $f \in S E Q$ (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 $h i=h d(f i)$ for $i$
show ?case
proof (cases $\exists i . f i=[])$
case False
then have $n e: \forall i . f i \neq[]$ by auto
with $\langle f \in S E Q$ (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 $f^{\prime}$ where $f^{\prime} i=($ if $i<\varphi 0$ then $f i$ else tl $(f(\varphi(i-\varphi 0)))$ ) for $i$
have $f^{\prime}: f^{\prime} \in S E Q$ (lists $A$ ) using $n e$ and $\langle f \in S E Q$ (lists $A$ ) $\rangle$
by (auto simp: $f^{\prime}$-def dest: list.set-sel)
have $[\operatorname{simp}]: \bigwedge i . \varphi 0 \leq i \Longrightarrow h(\varphi(i-\varphi 0)) \# f^{\prime} i=f(\varphi(i-\varphi 0))$
$\bigwedge i . i<\varphi 0 \Longrightarrow f^{\prime} i=f i$ using ne by (auto simp: $f^{\prime}$-def $h$-def)
moreover have strict-suffix ( $\left.f^{\prime}(\varphi 0)\right)(f(\varphi 0))$ using ne by (auto simp: $\left.f^{\prime}-d e f\right)$
ultimately have $L E X$ strict-suffix $f^{\prime} f$ by (auto simp: LEX-def)
with LEX-imp-not-LEX [OF this] have strict (LEXEQ strict-suffix) $f^{\prime} f$
using po-on-strict-suffix [of UNIV] unfolding po-on-def irreflp-on-def transp-on-def by blast
from less(2) $\left[O F f^{\prime}\right.$ this] have good (list-emb P) $f^{\prime}$.
then obtain $i j$ where $i<j$ and emb: list-emb $P\left(f^{\prime} i\right)\left(f^{\prime} j\right)$ by (auto simp: good-def)
consider $j<\varphi 0|\varphi 0 \leq i| i<\varphi 0$ and $\varphi 0 \leq j$ by arith
then show ?thesis
proof (cases)
case 1 with $\langle i<j\rangle$ and emb show ?thesis by (auto simp: good-def)
next
case 2
with $\langle i<j\rangle$ and $P$ have $P(h(\varphi(i-\varphi 0)))(h(\varphi(j-\varphi 0)))$ by auto
with emb have list-emb $P\left(h(\varphi(i-\varphi 0)) \# f^{\prime} i\right)\left(h(\varphi(j-\varphi 0)) \# f^{\prime}\right.$
j) by auto
then have list-emb $P(f(\varphi(i-\varphi 0)))(f(\varphi(j-\varphi 0)))$ using 2 and $\langle i$ $<j\rangle$ by auto
moreover with 2 and $\langle i<j\rangle$ 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 $P(f i)\left(f^{\prime} j\right)$ by auto
moreover have $f(\varphi(j-\varphi 0))=h(\varphi(j-\varphi 0)) \# f^{\prime} j$ using 3 by auto
ultimately have list-emb $P(f i)(f(\varphi(j-\varphi 0)))$ by auto

```
            moreover have i<\varphi (j-\varphi0) using mono [of 0j - \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 $m b s$ ) $m b s^{\prime}$ :
assumes $\neg$ almost-full-on $P A$
shows $\exists m \in B A D P . \forall g .(m, g) \in$ gseq $\longrightarrow \operatorname{good} P g$
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 (-」 [1000] 1000)
where

$$
A_{\perp}=\{N o n e\} \cup S o m e \text { ' } A
$$

lemma with-bot-iff [iff]:
Some $x \in A_{\perp} \longleftrightarrow x \in A$
by (auto simp: with-bot-def)
lemma NoneI [simp, intro]:
None $\in A_{\perp}$
by (simp add: with-bot-def)
lemma not-None-the-mem [simp]:
$x \neq$ None $\Longrightarrow$ the $x \in A \longleftrightarrow x \in A_{\perp}$
by auto
lemma with-bot-cases:
$u \in A_{\perp} \Longrightarrow(\bigwedge x . x \in A \Longrightarrow u=$ Some $x \Longrightarrow P) \Longrightarrow(u=$ None $\Longrightarrow P) \Longrightarrow P$
by auto
lemma with-bot-empty-conv [iff]:
$A_{\perp}=\{$ None $\} \longleftrightarrow A=\{ \}$
by (auto elim: with-bot-cases)
lemma with-bot-UNIV [simp]:
$U N I V_{\perp}=U N I V$
proof (rule set-eqI)
fix $x$ :: 'a option
show $x \in U N I V_{\perp} \longleftrightarrow x \in U N I V$ by (cases $x$ ) auto qed

### 7.2 Adding a Bottom Element to an Almost-Full Set

## fun

option-le :: ( ${ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool $) \Rightarrow{ }^{\prime} a$ option $\Rightarrow{ }^{\prime} a$ option $\Rightarrow$ bool
where
option-le $P$ None $y=$ True
option-le $P($ Some $x)$ None $=$ False $\mid$
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 $P$ ) $A_{\perp}$ (is almost-full-on ?P ?A)
proof
fix $f::$ nat $\Rightarrow{ }^{\prime}$ '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 $\langle i<j\rangle$ show good ?P $f$ by (auto simp: good-def)
qed auto
qed

### 7.3 Disjoint Union of Almost-Full Sets

fun
sum-le $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow\left({ }^{\prime} b \Rightarrow{ }^{\prime} b \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a+{ }^{\prime} b \Rightarrow{ }^{\prime} a+{ }^{\prime} b \Rightarrow$ bool
where

```
sum-le \(P Q(\) Inl \(x)(\) Inl \(y)=P x y \mid\)
sum-le \(P Q(\) Inr \(x)(\) Inr \(y)=Q x y \mid\)
sum-le \(P Q x y=\) False
lemma not-sum-le-cases:
    assumes \(\neg\) sum-le \(P Q a b\)
    and \(\bigwedge x y . \llbracket a=\operatorname{Inl} x ; b=\operatorname{Inl} y ; \neg P x y \rrbracket \Longrightarrow\) thesis
    and \(\bigwedge x y . \llbracket a=\operatorname{Inr} x ; b=\operatorname{Inr} y ; \neg Q x y \rrbracket \Longrightarrow\) thesis
```

> and $\bigwedge x y \cdot \llbracket a=\operatorname{Inl} x ; b=\operatorname{Inr} y \rrbracket \Longrightarrow$ thesis and $\bigwedge x y \cdot \llbracket a=\operatorname{Inr} x ; b=\operatorname{Inl} y \rrbracket \Longrightarrow$ 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 $\Rightarrow\left({ }^{\prime} a+{ }^{\prime} b\right)$
let ? $I=f-{ }^{\prime}$ Inl' $A$
let ? $J=f-{ }^{\prime}$ 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 $\lambda i . f i \in \operatorname{Inr}$ ' $B$
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 $=\lambda i . \operatorname{projr}(f($ 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 $\lambda i . f i \in \operatorname{Inl}$ ' $A$ 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 \negP(?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 :: (' \(a \Rightarrow{ }^{\prime} a \Rightarrow\) bool \() \Rightarrow\left({ }^{\prime} b \Rightarrow{ }^{\prime} b \Rightarrow\right.\) bool \() \Rightarrow{ }^{\prime} a \times{ }^{\prime} b \Rightarrow^{\prime} a \times{ }^{\prime} b \Rightarrow\) bool
where
    prod-le P1 P2 \(=(\lambda(p 1, p 2)(q 1, q 2)\). P1 p1 q1 \(\wedge\) P2 \(p 2 q 2)\)
lemma prod-le-True [simp]:
    prod-le \(P(\lambda\) - -. True) \(a b=P(f s t a)(f s t b)\)
    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 \(\left(f_{s t} x\right)\left(f_{s t} y\right)\)
    let \(? B=\lambda x y . P 2(\) snd \(x)(\) snd \(y)\)
    from \(f\) have \(f s t: \forall i\). fst \((f i) \in A 1\) and snd: \(\forall i\). snd \((f i) \in A 2\)
        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 \(\circ f \circ \varphi \in S E Q\) 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 \(*[O F\langle i<j\rangle]\) have ? \(P(f(\varphi i))(f(\varphi j))\) by (simp add: case-prod-beta
prod-le-def)
    with mono \([O F \prec i<j\rangle]\) 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 \neg? ?thesis
    from mbs' [OF this] obtain m
        where bad:m}\inBAD?
        and min: }\forallg.(m,g)\ingseq\longrightarrowgood ?P g ..
    then have lists: \i.mi\inlists A
        and ne: \i.m i\not= [] by auto
```

    define \(h t\) where \(h=(\lambda i . h d(m i))\) and \(t=(\lambda i . t l(m i))\)
    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 \([O F n e]\) and lists by (auto simp add: \(h\)-def)
    from almost-full-on-imp-homogeneous-subseq \([O F\) 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 $\langle i<j\rangle$ have good?P $m$ by (auto simp: good-def)
with bad show False by blast
qed
define $m^{\prime}$ where $m^{\prime}=(\lambda i$. if $i<\varphi 0$ then $m$ i else $t(\varphi(i-\varphi 0)))$
have $m^{\prime}$-less: $\bigwedge i . i<\varphi 0 \Longrightarrow m^{\prime} i=m i$ by (simp add: $m^{\prime}$-def)
have $m^{\prime}$-geq: $\bigwedge i . i \geq \varphi 0 \Longrightarrow m^{\prime} i=t(\varphi(i-\varphi 0))$ by (simp add: $m^{\prime}$-def)
have $\forall i . m^{\prime} i \in$ lists $A$ using ne-lists $[O F n e]$ and lists by (auto simp: $m^{\prime}$-def
$t$-def)
moreover have length $\left(m^{\prime}(\varphi 0)\right)<$ length $(m(\varphi 0))$ using ne by (simp add:
$t$-def $m^{\prime}$-geq)
moreover have $\forall j<\varphi 0 . m^{\prime} j=m j$ by (auto simp: $m^{\prime}$-less)
ultimately have $\left(m, m^{\prime}\right) \in$ gseq using lists by (auto simp: gseq-def)
moreover have bad ?P $\mathrm{m}^{\prime}$
proof
assume good ? P $\mathrm{m}^{\prime}$
then obtain $i j$ where $i<j$ and emb: ?P $\left(m^{\prime} i\right)\left(m^{\prime} j\right)$ by (auto simp:
good-def)

```
    { assume j<\varphi0
        with }\langlei\langlej\rangle\mathrm{ and emb have ?P (mi) (mj) by (auto simp: m'-less)
        with }\langlei<j\rangle\mathrm{ and bad have False by blast }
    moreover
    { assume \varphi 0 \leqi
        with \langlei< j> and emb have ?P (t (\varphi (i-\varphi 0))) (t (\varphi (j - \varphi 0)))
            and i-\varphi0<j-\varphi0 by (auto simp: m'-geq)
    with bad-t have False by auto }
    moreover
    { assume i< \varphi 0 and \varphi 0\leqj
    with <i<j> and emb have ?P (m i) (t (\varphi (j - \varphi 0))) by (simp add: m'-less
m'-geq)
            from list-emb-Cons [OF this, of h (\varphi (j-\varphi 0))]
                have ?P (m i) (m (\varphi (j - \varphi0))) using ne by (simp add: h-def t-def)
            moreover have i<\varphi(j-\varphi0)
                using less [of 0j - \varphi 0] and <i<\varphi 0\rangle and <\varphi 0\leqj\rangle
                by (cases j=\varphi 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 (\leq) (UNIV :: nat set)
proof -
    let ?P = subseq :: bool list }=>\mathrm{ bool list }=>\mathrm{ bool
    have *: length '(UNIV :: bool list set) = (UNIV :: nat set)
        by (metis Ex-list-of-length surj-def)
    have almost-full-on (\leq) (length '(UNIV :: bool list set))
    proof (rule almost-full-on-hom)
        fix xs ys :: bool list
        assume ?P xs ys
        then show length xs \leq 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<br>imports Almost-Full-Relations<br>begin

### 8.1 Basic Definitions

definition wqo-on :: ('a $\Rightarrow^{\prime} a \Rightarrow$ bool $) \Rightarrow{ }^{\prime}$ 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 ( $\lambda$ - -. True) UNIV
using almost-full-on-UNIV by (auto simp: wqo-on-def transp-on-def)
lemma wqo-onI [Pure.intro]:
$\llbracket t r a n s p-o n A P$; almost-full-on $P A \rrbracket \Longrightarrow$ 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 $P A \Longrightarrow q o-o n ~ P A$
by (metis qo-on-def wqo-on-imp-reflp-on wqo-on-imp-transp-on)
lemma wqo-on-imp-good:
wqo-on $P A \Longrightarrow \forall i . f i \in A \Longrightarrow \operatorname{good} P f$
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 \longleftrightarrowwfp-on (strict P) A ^\neg(\existsf. antichain-on Pf 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 ( 
\longrightarrow w f p - o n ~ ( s t r i c t ~ 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 \Longrightarrowwfp-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 \(\quad\) Q h]
    unfolding wqo-on-def by blast
```

The monomorphic preimage of a wqo set is wqo.

```
lemma wqo-on-mon:
    assumes *: \forallx\inA.\forally\inA.P }P\mathrm{ x }\longleftrightarrow\longleftrightarrowQ(hx)(hy
    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 }xyz\mathrm{ assume [intro!]: }x\inAy\inAz\in
                and Pxy and Pyz
        with * have Q (hx) (hy) and Q (hy) (hz) by blast+
        with wqo-on-imp-transp-on [OF wqo] have Q (hx) (hz)
            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 (\leq)f
lemma wqo-on-class [simp, intro]:
    wqo-on (\leq) (UNIV :: ('a :: wqo) set)
    using good by (auto simp: wqo-on-def transp-on-def almost-full-on-def dest: or-
der-trans)
lemma wqo-on-UNIV-class-wqo [intro!]:
    wqo-on P UNIV \Longrightarrow 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 $\longleftrightarrow$ 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 \((\leq)(<)\)..
    hence wqo-on ( \(\leq\) ) 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 $P$ ) $A_{\perp}$ (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 $A 1$ :: ' $a$ set and $A 2$ :: ' $b$ 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 \Longrightarrow wqo-on Q UNIV \Longrightarrow 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 \Longrightarrow 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 $P^{\prime} \equiv \lambda x y . P x y \wedge Q(h x)(h y)$
assumes wqo－on $P A$
and wqo－on $Q B$
and subset：$h$＇$A \subseteq B$
shows wqo－on $P^{\prime} A$
proof
let $? Q=\lambda x y . Q(h x)(h y)$
from $\langle$ wqo－on $P A$ 〉 have transp－on $A P$
by（rule wqo－on－imp－transp－on）
then show transp－on $A P^{\prime}$
using 〈wqo－on $Q B$ 〉 and subset
unfolding wqo－on－def transp－on－def $P^{\prime}$－def by blast
from 〈wqo－on $P A$ 〉 have almost－full－on $P A$
by（rule wqo－on－imp－almost－full－on）
from $\langle$ wqo－on $Q B$ 〉 have almost－full－on $Q B$
by（rule wqo－on－imp－almost－full－on）
show almost－full－on $P^{\prime} A$
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 $* *\langle$ transp－on A $P\rangle$ ］
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 $* *[O F<i<j\rangle]$ have $P^{\prime}(f(g i))(f(g j))$
by（auto simp：$P^{\prime}$－def）
with $g[O F\langle i<j\rangle]$ show good $P^{\prime} f$ by（auto simp：good－def）
qed
qed
lemma wqo－on－UNIV－nat：

```
wqo-on (\leq) (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 $F::\left({ }^{\prime} b \times n a t\right)$ set and $m k:: ' b \Rightarrow$ '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 $t s) \in F \Longrightarrow \operatorname{root}(m k f t s)=(f$, length $t s)$ and args-mk: $(f$, length $t s) \in F \Longrightarrow$ args $(m k f t s)=t s$ and $m k$-root-args: $t \in$ trees $\Longrightarrow m k(f s t($ 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 $t s) \in F$
shows $m k f s s=m k g t s \longleftrightarrow f=g \wedge s s=t s$
proof -
\{ assume $m k f s s=m k g t s$
then have root $(m k f s s)=\operatorname{root}(m k g t s)$
and args ( $m k f s s$ ) $=\operatorname{args}(m k g t s)$ by auto $\}$
show ?thesis
using root-mk $[O F \operatorname{assms}(1)]$ and root-mk $[O F \operatorname{assms}(2)]$
and args-mk $[O F \operatorname{assms}(1)]$ and $\operatorname{args}-m k[O F \operatorname{assms}(2)]$ by auto
qed
inductive $e m b$ for $P$
where
arg: $\llbracket(f, m) \in F ;$ length $t s=m ; \forall t \in$ set ts. $t \in$ trees; $t \in s e t t s ; e m b P s t \rrbracket \Longrightarrow e m b P s(m k f t s) \mid$
list-emb: $\llbracket(f, m) \in F ;(g, n) \in F ;$ length $s s=m$; length $t s=n$; $\forall s \in$ set ss. $s \in$ trees $; \forall t \in$ set $t s . t \in$ trees; $P(f, m)(g, n) ;$ list-emb $(e m b P) s s t s \rrbracket \Longrightarrow e m b P(m k f s s)(m k g t s)$
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 \neg ?thesis
    from mbs'[OF this] obtain m
        where bad:m}\inBAD ?P
        and min: }\forallg.(m,g)\ingseq\longrightarrowgood?P g ..
    then have trees: \i.mi\in trees by auto
    define r where ri= root (mi) for i
define a where a i= args (mi) for i
define S where S=\bigcup{set (ai)|i.True}
have m: \i.mi=mk(fst (ri)) (a i)
    by (simp add: r-def a-def mk-root-args [OF trees])
have lists: \foralli. a i lists S by (auto simp: a-def S-def)
have arity: \i. length (a i)= snd (ri)
    using trees-arity [OF trees] by (auto simp: r-def a-def)
then have sig: \bigwedgei. (fst (r i), length (a i)) \inF
    using trees-root [OF trees] by (auto simp: a-def r-def)
have a-trees: \bigwedgei.\forallt\in set (a i).t\in trees by (auto simp: a-def trees-args [OF
trees])
have almost-full-on ?P S
proof (rule ccontr)
    assume \neg ?thesis
    then obtain s:: nat }\mp@subsup{=>}{}{\prime}
        where S: \bigwedgei. s i\inS and bad-s: bad ?P s by (auto simp: almost-full-on-def)
    define }n\mathrm{ where n=(LEAST n. }\exists\textrm{k}.\textrm{s}k\in\operatorname{set}(an)
    have }\existsn.\existsk.sk\in\operatorname{set (a n) using S by (force simp:S-def)
    from LeastI-ex [OF this] obtain k
        where sk: sk\in set (a n) by (auto simp: n-def)
    have args: \bigwedgek.\existsm\geqn.sk set (am)
        using}S\mathrm{ by (auto simp: S-def) (metis Least-le n-def)
    define m' where m'
    have m'-less: \i. i<n\Longrightarrow m'i=m i by (simp add: m'-def)
    have m'-geq: \bigwedgei. i\geqn\Longrightarrow m'i}=s(k+(i-n)) by (simp add: m'-def
    have bad ?P m'
    proof
        assume good ?P m'
        then obtain ij where i<j and emb:?P (m'i) ( m'j) by auto
        { assume j<n
            with }\langlei<j\rangle\mathrm{ and emb have ?P (m i) (mj) by (auto simp: m'-less)
            with }\langlei<j\rangle\mathrm{ and bad have False by blast }
```

```
    moreover
    { assume n \leqi
        with <i<j\rangle 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\leqj
        with <i< j\rangle and emb have *: ?P (mi) (s (k+(j-n))) by (auto simp:
m'-less m'-geq)
    with args obtain l where l\geqn 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\mathrm{ using < }i<n\rangle\mathrm{ and <n }\leql\rangle\mathrm{ by auto
            ultimately have False using bad by blast }
            ultimately show False using < < < j` by arith
    qed
    moreover have (m, m')\ingseq
    proof -
        have m}\inSEQ ?A using trees by aut
        moreover have m' }\inSEQ\mathrm{ ?A
            using trees and S and trees-args [OF trees] by (auto simp: m'-def a-def
S-def)
            moreover have }\foralli<n.mi=\mp@subsup{m}{}{\prime}i\mathrm{ 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 \varphi :: nat => nat
    where less: \ij. i<j\Longrightarrow\varphii<\varphij
    and lemb: \bigwedgeij.i<j\Longrightarrow list-emb ?P (a (\varphi i)) (a (\varphij)) by blast
    have roots: \i.r (\varphi i)\inF using trees [THEN trees-root] by (auto simp: r-def)
    then have }r\circ\varphi\inSEQF\mathrm{ by auto
    with assms have good P (r\circ\varphi) by (auto simp: almost-full-on-def)
    then obtain ij
    where i<j and P(r(\varphii)) (r (\varphij)) 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\rangle] 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 (\lambdayz.Pyz\wedge(\forallx.Pxy\longrightarrowPxz)) 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: \fgh.f\inF\Longrightarrowg\inF\Longrightarrowh\inF\LongrightarrowPfg\LongrightarrowPgh\LongrightarrowP
f h
    assumes emb P st and emb Ptu
    shows emb P s u
using assms(3, 2)
proof (induct arbitrary:s)
    case (arg fmtsv)
    then show ?case by (auto intro: emb.arg)
next
    case (list-emb fmg n ss ts)
    note IH = this
    from <emb Ps(mkfss)>
        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 $f t s)=(f$, length ts $)$
fun succs
where
succs $($ Node $f t s)=t s$
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 $U N I V=U N I V$
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 $A \times$ 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 $P(\lambda-$-. 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 $s \leq t \longleftrightarrow$ tree-emb UNIV $(\leq) s t$

```
definition (s :: 'a tree ) <t \longleftrightarrow s\leqt\wedge\neg (t\leqs)
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 (Funfts)=(f, length ts)
fun args
where
    args (Funfts)=ts
inductive-set gterms for F
where
    (f,n) \inF\Longrightarrow length ts = n\Longrightarrow \Longrightarrows set ts.s\in gterms F\Longrightarrow Fun fts }\in\mathrm{ 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
    \forallt\in set ts. t\in terms \LongrightarrowFun fts\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(vx) [] = V x 
    mk (cn) [] = Cn |
    mk p[a,b]=Plus a b
```


## fun $r t$

```
where
```

```
rt (V x) = (v x, 0::nat) |
```

rt (V x) = (v x, 0::nat) |
rt (C n) = (c n,0)|
rt (C n) = (c n,0)|
rt (Plus a b)=(p, 2)

```
rt (Plus a b)=(p, 2)
```

fun ags
where
ags $(V x)=[] \mid$
ags $(C n)=[] \mid$
ags $($ Plus a b) $=[a, b]$
inductive-set exps
where

```
    V x \in exps 
Cn\inexps
a\in exps \Longrightarrowb\in exps \Longrightarrow Plus a b exps
```

lemma [simp]:
assumes length $t s=2$
shows $r t(m k p t s)=(p, 2)$
using assms by (induct ts) (auto, case-tac ts, auto)
lemma [simp]:
assumes length ts $=2$
shows ags ( $m k p t s$ ) $=t s$
using assms by (induct ts) (auto, case-tac ts, auto)
interpretation kruskal-exp: kruskal-tree
$\{(v x, 0) \mid x$. True $\} \cup\{(c n, 0) \mid n$. True $\} \cup\{(p, 2)\}$
$m k$ 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 $x \leq y \longleftrightarrow$ option-le $(\leq) x y$
definition $(x:: ' a$ option $)<y \longleftrightarrow x \leq y \wedge \neg(y \leq x)$
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\leqy\longleftrightarrow\mathrm{ sum-le (土)( }\leq\mathrm{ ) x y
definition (x::' }a+\mp@subsup{}{}{\prime}b)<y\longleftrightarrowx\leqy\wedge\neg(y\leqx
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 $p \leq q \longleftrightarrow$ prod-le $(\leq)(\leq) p q$
definition $\left(p::{ }^{\prime} a \times{ }^{\prime} b\right)<q \longleftrightarrow p \leq q \wedge \neg(q \leq p)$
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\leqys\longleftrightarrowlist-emb (\leq) xs ys
definition (xs :: 'a list) < ys \longleftrightarrowxs\leqys\wedge\neg(ys\leqxs)
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-mset $M \subseteq A\}$
lemma in-multisets-iff:
$M \in$ multisets $A \longleftrightarrow$ set-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 :: (' $a \Rightarrow{ }^{\prime} a \Rightarrow$ bool $) \Rightarrow{ }^{\prime} a$ multiset $\Rightarrow{ }^{\prime} a$ 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 $N(M 0+\{\# a \#\}) \Longrightarrow$
$(\exists M$. mulex1 $P M M 0 \wedge N=M+\{\# a \#\}) \vee$
$(\exists K .(\forall b . b \in \# K \longrightarrow P b a) \wedge N=M 0+K)$
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 sim
    moreover have }A=A+{#}\mathrm{ by simp
    moreover have }\forallb.b\in#{#}\longrightarrow(b,a)\in?R by sim
    ultimately have ( }A\mathrm{ , add-mset a A) G 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 }\forallb.b\in#{#}\longrightarrow(b,a)\inR by sim
    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=>\mp@subsup{|}{}{\prime}a=>\mathrm{ bool ) }=>\mathrm{ ' 'a set }=>\mp@subsup{|}{}{\prime}a\mathrm{ multiset }=>\mp@subsup{}{}{\prime}a\mathrm{ multiset }
bool where
    mulex-on P A = (restrict-to (mulex1 P) (multisets A))}\mp@subsup{)}{}{++
abbreviation mulex ::(' }a=>\mp@subsup{|}{}{\prime}a=>\mathrm{ bool ) }=>\mp@subsup{'}{}{\prime}a\mathrm{ multiset }=>\mp@subsup{}{}{\prime}\a multiset => boo
where
    mulex P}\equiv\mathrm{ 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 }\bigwedgeMN.\llbracketM\in\mathrm{ multisets A;N E multisets A; mulex1 P M N】 בQ MN
    and \LMN.\llbracketmulex-on P A L M;QLM;N\in multisets A; mulex1 P M N\rrbracket
QLN
    shows QMN
    using assms unfolding mulex-on-def by (induct) blast+
lemma mulex-on-self-add-singleton-right [simp]:
    assumes }a\inA\mathrm{ and M multisets }
    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]
$\{\# x \#\} \in$ multisets $A \longleftrightarrow x \in A$ by (auto simp: multisets-def)
lemma union-multisetsD:
assumes $M+N \in$ multisets $A$ shows $M \in$ multisets $A \wedge N \in$ multisets $A$ using assms by (auto simp: multisets-def)
lemma mulex-on-multisetsD [dest]:
assumes mulex-on P F M N
shows $M \in$ multisets $F$ and $N \in$ multisets $F$
using assms by (induct) auto
lemma union-multisets-iff [iff]:
$M+N \in$ multisets $A \longleftrightarrow M \in$ multisets $A \wedge N \in$ multisets $A$
by (auto dest: union-multisetsD)
lemma add-mset-multisets-iff [iff]:
add-mset $a M \in$ multisets $A \longleftrightarrow a \in A \wedge M \in$ multisets $A$
unfolding add-mset-add-single[of a $M$ ] union-multisets-iff by auto
lemma mulex-on-trans:
mulex-on $P A L M \Longrightarrow$ mulex-on $P A M N \Longrightarrow$ 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 PA] by (auto simp: transp-on-def)
lemma mulex-on-add-right [simp]:
assumes mulex-on $P A M N$ and $a \in A$
shows mulex-on P A M (add-mset a $N$ )
proof -
from assms have $a \in A$ and $N \in$ 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 $M \neq\{\#\}$ and $M \in$ 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\not={#}
        with add show ?thesis by (auto intro: mulex-on-trans)
    qed
qed simp
lemma mulex-on-self-add-right [simp]:
    assumes }M\in\mathrm{ multisets }A\mathrm{ and }K\in\mathrm{ multisets A and }K\not={#
    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\not={#} with add show ?thesis
            by (auto dest: mulex-on-add-right simp add: ac-simps)
    qed
qed
lemma mult1-singleton [iff]:
    ({#x#},{#y#})\in mult1 R}\longleftrightarrow(x,y)\in
proof
    assume (x,y)\inR
    then have {#y#}={#}+{#y#}
        and {#x#}={#}+{#x#}
        and }\forallb,b\in#{#x#}\longrightarrow(b,y)\inR\mathrm{ by auto
    then show ({#x#},{#y#})\in mult1 R unfolding mult1-def by blast
next
    assume ({#x#},{#y#}) \in mult1 R
    then obtain MO K a
        where {#y#} = add-mset a M0
        and {#x#} = M0 + K
        and }\forallb.b\in#K\longrightarrow(b,a)\in
        unfolding mult1-def by blast
    then show (x,y)\inR by (auto simp: add-eq-conv-diff)
qed
lemma mulex1-singleton [iff]:
    mulex1 P {#x#} {#y#}\longleftrightarrowP 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 $(\text { mulex-on } P A)^{=}=M N$ and $M \in$ 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 $(\text { mulex-on } P A)^{=}=M N$ and $M \in$ multisets $A$ and $a \in A$
shows mulex-on PAM(\{\#a\#\}+N)
using reffclp-mulex-on-add-right [OF assms] by (simp add: ac-simps)
lemma mulex-on-union-right [simp]:
assumes mulex-on PFAB and $K \in$ 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 FA $((B+K)+\{\# a \#\})$ by simp
then show ?case by (simp add: ac-simps)
qed simp
lemma mulex-on-union-right' ${ }^{[\text {simp }]:}$
assumes mulex-on PFAB and $K \in$ multisets $F$
shows mulex-on P FA $(B+K)$
using mulex-on-union-right [OF assms] by (simp add: ac-simps)
Adapted from $w f ? r \Longrightarrow \forall M . M \in$ Wellfounded.acc (mult1 ?r) in HOL-Library.Multiset.
lemma accessible-on-mulex1-multisets:
assumes wf: wfp-on P A
shows $\forall M \in$ multisets $A$. accessible-on (mulex1 P) (multisets A) M

## proof

let $? P=$ mulex $1 P$
let $? A=$ multisets $A$
let ?acc $=$ accessible-on $? P$ ? $A$
\{
fix $M$ M0 a
assume M0: ?acc M0
and $a \in A$
and $M 0 \in ? A$
and wf-hyp: $\wedge b . \llbracket b \in A ; P b a \rrbracket \Longrightarrow(\forall M$. ?acc $(M) \longrightarrow$ ?acc $(M+\{\# b \#\}))$
and acc-hyp: $\forall M . M \in ? A \wedge ? P M M 0 \longrightarrow$ ?acc $(M+\{\# a \#\})$
then have add-mset a $M 0 \in ? A$ by (auto simp: multisets-def)

```
then have ?acc (add-mset a M0)
proof (rule accessible-onI [of add-mset a M0])
    fix }
    assume N\in?A
        and ?P N (add-mset a M0)
    then have ((\existsM.M\in?A\wedge?P M M0 ^N=M + {#a#}) \vee
            (\existsK.(\forallb.b\in#K\longrightarrowPba)\wedgeN=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 \in?A ^?P M M0 \longrightarrow?acc (M+{#a#})..
        with }\langleM\in?A\\mathrm{ and <?P M M0` have ?acc (M + {#a#}) by blast
        then show ?acc (N) by (simp only:N)
    next
        fix }
        assume N:N=M0 + K
        assume }\forallb.b\in#K\longrightarrowPb
    moreover from N and }\langleN\in?A\rangle\mathrm{ have K}\in?A\mathrm{ 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\inA and P x a by (auto simp: multisets-def)
            with wf-hyp have }\forallM\mathrm{ . ?acc }M\longrightarrow\mathrm{ ?acc ( }M+{#x#})\mathrm{ 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\in?A
then show ?acc M
proof (induct M)
    show ?acc {#}
    proof (rule accessible-onI)
        show {#} \in ?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\inA by (auto simp: multisets-def)
```

```
    with wf have }\forallM\mathrm{ . ?acc M }\longrightarrow\mathrm{ ?acc (add-mset a M)
```

    proof (induct)
    case (less a)
    then have \(r: \wedge b . \llbracket b \in A ; P b a \rrbracket \Longrightarrow(\forall M\). ?acc \(M \longrightarrow\) ?acc \((M+\{\# b \#\}))\)
    by auto
show $\forall M$. ?acc $M \longrightarrow$ ? acc (add-mset a $M$ )
proof (intro allI impI)
fix $M^{\prime}$
assume ?acc $M^{\prime}$
moreover then have $M^{\prime} \in ? A$ by (blast dest: accessible-on-imp-mem)
ultimately show ?acc (add-mset a $M^{\prime}$ )
by (induct) (rule tedious-reasoning $[O F-\langle a \in A\rangle-r]$, auto)
qed
qed
with «?acc (M)〉 show ?acc (add-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-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-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))}\mp@subsup{)}{}{++}(K+M)(K+N
by auto
    moreover have mulex-on PA (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 PAMN 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 PAMN and m}\in
    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 Pmn and m\inA and n\inA 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 Pmn and m\inA and n\inA and mulex-on P A MN
    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 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 $\bigwedge x y . \llbracket x \in A ; y \in A ;(x, y) \in R \rrbracket \Longrightarrow(x, y) \in S$
and $M \in$ multisets $A$
and $N \in$ multisets $A$
and $(M, N) \in$ mult1 $R$
shows $(M, N) \in$ mult $1 S$
using assms unfolding mult1-def multisets-def
by auto (metis (full-types) subsetD)
lemma mulex1-mono:
assumes $\bigwedge x y . \llbracket x \in A ; y \in A ; P x y \rrbracket \Longrightarrow Q x y$
and $M \in$ multisets $A$
and $N \in$ 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 $*: \bigwedge x y . \llbracket x \in A ; y \in A ; P x y \rrbracket \Longrightarrow Q x y$
and mulex-on $P A M N$
shows mulex-on $Q A M N$
proof -
let ?rel $=\lambda P$. (restrict-to (mulex1 P) (multisets A))
from $\langle m u l e x-o n P A M N \text { ไ have (? rel } P)^{++} M N$ by (simp add: mulex-on-def)
then have $(\text { ? rel } Q)^{++} M N$
proof (induct rule: tranclp.induct)
case ( $r$-into-trancl $M N$ )
then have $M \in$ multisets $A$ and $N \in$ multisets $A$ by auto
from mulex1-mono $[O F *$ this $]$ and $r$-into-trancl
show ?case by auto

## next

case (trancl-into-trancl LMN)
then have $M \in$ multisets $A$ and $N \in$ multisets $A$ by auto
from mulex1-mono [OF * this] and trancl-into-trancl
have ? $\mathrm{rel} Q M N$ by auto
with $\left\langle(\text { ?rel } Q)^{++} L M\right\rangle$ show ?case by (rule tranclp.trancl-into-trancl)
qed
then show?thesis by (simp add: mulex-on-def)
qed
lemma mult1-reflcl:
assumes $(M, N) \in$ mult1 $R$
shows $(M, N) \in$ mult1 $\left(R^{=}\right)$
using assms by (auto simp: mult1-def)
lemma mulex1-reflclp:

```
    assumes mulex1 P M N
    shows mulex1 ( }P===)M
    using mulex1-mono [of UNIV P P}===MN,OF -- assms
    by (auto simp: multisets-def)
lemma mulex-on-reflclp:
    assumes mulex-on P A M N
    shows mulex-on ( }P==\mathrm{ ) AMN
    using mulex-on-mono [OF - assms, of P==] by auto
lemma surj-on-multisets-mset:
    \forallM\inmultisets A. \existsxs\inlists A. M = mset xs
proof
    fix M
    assume M E multisets A
    then show \existsxs\inlists 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 \in lists A using <xs \in 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\not={#}
        and \\x.P{#x#}
        and }\xM.PM\LongrightarrowP(add-mset x M
    shows P M
    using assms by (induct M) auto
lemma mulex-on-all-strict:
    assumes X\not={#}
    assumes }X\in\mathrm{ multisets }A\mathrm{ and }Y\in\mathrm{ multisets }
        and *: }\forally.y\in#Y\longrightarrow(\existsx.x\in#X\wedgePyx
    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. \existsx. x\in# M\wedgeP y x#}
    let ?Z = Y - ? Y
    have Y:Y=?Z + ?Y by (subst multiset-eq-iff) auto
    from \langleY\in multisets A\rangle have ? Y multisets A by (metis multiset-partition
union-multisets-iff)
    moreover have }\forally.y\in#?Y\longrightarrow(\existsx.x\in#M\wedgePyx) by aut
    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 }\forally.y\in#?Z\longrightarrowP y 
        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 }\langleY\in\mathrm{ 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 PA(?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 multisetextension 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 \(a M 0 K\) where \(N: N=M 0+\{\# a \#\}\)
            and \(M: M=M 0+K\)
            and \(*: \forall b . b \in \# K \longrightarrow P b a\)
            and \(M \in\) multisets \(A\) and \(N \in\) multisets \(A\) by (auto simp: mulex1-def
mult1-def)
    moreover then have \(\{\# a \#\} \in\) multisets \(A\) and \(K \in\) multisets \(A\) by auto
    moreover have \(\{\# a \#\} \neq\{\#\}\) by auto
    moreover have \(N=M 0+\{\# a \#\}\) by fact
    moreover have \(M=M 0+K\) by fact
        moreover have \(\forall y . y \in \# K \longrightarrow(\exists x . x \in \#\{\# a \#\} \wedge P y x)\) using \(*\) by
auto
    ultimately show ?case by blast
    next
        case (step L M N)
        then obtain \(X Y Z\)
            where \(L \in\) multisets \(A\) and \(M \in\) multisets \(A\) and \(N \in\) multisets \(A\)
            and \(X \in\) multisets \(A\) and \(Y \in\) multisets \(A\)
            and \(M: M=Z+X\)
            and \(L: L=Z+Y\) and \(X \neq\{\#\}\)
            and \(Y: \forall y . y \in \# Y \longrightarrow(\exists x . x \in \# X \wedge P y x)\)
            and mulex1 \(P M N\)
            by blast
            from 〈mulex1 \(P M N\rangle\) obtain a M0 \(K\)
            where \(N: N=\) add-mset a \(M 0\) and \(M^{\prime}: M=M 0+K\)
            and \(*: \forall b . b \in \# K \longrightarrow P b\) a unfolding mulex1-def mult1-def by blast
    have \(L^{\prime}: L=(M-X)+Y\) by \((\operatorname{simp}\) add: \(L M)\)
    have \(K: \forall y . y \in \# K \longrightarrow(\exists x . x \in \#\{\# a \#\} \wedge P y x)\) 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\in\mathrm{ multisets A and N}\in\mathrm{ multisets A by fact+
moreover have ? }X\not={#}\wedge(\existsZ.N=Z+?X\wedgeL=Z + ?Y
proof -
    have ?X # { {#} by auto
    moreover have ?X \subseteq#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)+?
    proof (rule multiset-eqI)
        fix x :: 'a
        let ?c = \lambdaM. count M x
        let ?ic = \lambdax. int (?c x)
```

```
    from \(\langle ? X \subseteq \# N\rangle\) 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 \((N-? X+? Y)=\operatorname{int}(? c N-? c\) ? \(X)+\) ?ic ? \(Y\) by simp
    also have \(\ldots=\operatorname{int}(? c N-(? c\{\# a \#\}+? c(X-K)))+\) ? \(i c(K-X)\)
    + ?ic $Y$ by simp
also have $\ldots=$ ? ic $N-($ ?ic $\{\# a \#\}+$ ? ic $(X-K))+$ ?ic $(K-X)+$
?ic $Y$
using of-nat-diff $[O F *]$ 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 $(\operatorname{simp}$ add: $N)$
also have $\ldots=$ ? ic $L$
unfolding $L^{\prime} M^{\prime} N$
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 $\langle P y x\rangle$ have ?thesis by auto \}
moreover
\{ assume $x \in \# K$ with $*$ have $P x$ a by auto
moreover have $y \in A$ using $\langle Y \in$ multisets $A\rangle$ and $\langle y \in \# Y\rangle$ by (auto
simp: multisets-def)
moreover have $a \in A$ using $\langle N \in$ multisets $A\rangle$ by (auto simp: $N$ )
moreover have $x \in A$ using $\langle M \in$ multisets $A\rangle$ and $\langle x \in \# K\rangle$ by (auto
simp: $M^{\prime}$ multisets-def)
ultimately have $P$ y $a$ using $\langle P y x\rangle$ and trans unfolding transp-on-def
by blast
then have ?thesis by force \}
moreover from $\langle x \in \# X\rangle$ 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 XYZ where M multisets A and N\in multisets A
        and }X\not={#}\mathrm{ and N:N=Z+X and M:M=Z+Y
        and *:\forally.y\in#Y\longrightarrow(\existsx. x\in#X\wedgePyx) by blast
    with mulex-on-all-strict [of X A Y] have mulex-on P A Y X by auto
    moreover from }\langleN\in\mathrm{ multisets }A\rangle\mathrm{ have Z G 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 \(x s \in\) lists \(A\) and \(y s \in\) lists \(A\)
        and list-emb \(P\) xs ys
    shows (mulex-on \(P A\) ) \(==(\) mset \(x s)(\) mset \(y s)\)
using \(\operatorname{assms}(3,1,2)\)
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: multi-
sets-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 (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.


[^0]:    ${ }^{*}$ The research was funded by the Austrian Science Fund (FWF): J3202.

