

Well-Quasi-Orders

Christian Sternagel*

February 6, 2026

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 transitive relation	4
3	Enumerations of Well-Ordered Sets in Increasing Order	6
4	The Almost-Full Property	7
4.1	Basic Definitions and Facts	7
4.2	An equivalent inductive definition	8
4.3	Special Case: Finite Sets	11
4.4	Further Results	11
5	Constructing Minimal Bad Sequences	12

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

6	A Proof of Higman’s Lemma via Open Induction	15
6.1	Some facts about the suffix relation	15
6.2	Lexicographic Order on Infinite Sequences	15
7	Almost-Full Relations	17
7.1	Adding a Bottom Element to a Set	17
7.2	Adding a Bottom Element to an Almost-Full Set	17
7.3	Disjoint Union of Almost-Full Sets	18
7.4	Dickson’s Lemma for Almost-Full Relations	18
7.5	Higman’s Lemma for Almost-Full Relations	19
7.6	Natural Numbers	19
8	Well-Quasi-Orders	19
8.1	Basic Definitions	19
8.2	Equivalent Definitions	20
8.3	A Type Class for Well-Quasi-Orders	21
8.4	Dickson’s Lemma	22
8.5	Higman’s Lemma	22
9	Kruskal’s Tree Theorem	23
10	Instances of Well-Quasi-Orders	27
10.1	The Option Type is Well-Quasi-Ordered	27
10.2	The Sum Type is Well-Quasi-Ordered	27
10.3	Pairs are Well-Quasi-Ordered	27
10.4	Lists are Well-Quasi-Ordered	28
11	Multiset Extension of Orders (as Binary Predicates)	28
12	Multiset Extension Preserves Well-Quasi-Orders	34

1 Infinite Sequences

Some useful constructions on and facts about infinite sequences.

```

theory Infinite-Sequences
imports Main
begin

```

The set of all infinite sequences over elements from A .

```

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

```

```

lemma SEQ-iff [iff]:
   $f \in \text{SEQ } A \iff (\forall i. f\ i \in A)$ 
<proof>

```

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*]:

$f \in B \implies f\ i = x \implies x \in ith\ B\ i$
 ⟨*proof*⟩

lemma $ithE$ [*elim*]:

$\llbracket x \in ith\ B\ i; \bigwedge f. \llbracket f \in B; f\ i = x \rrbracket \implies Q \rrbracket \implies Q$
 ⟨*proof*⟩

lemma $ith\ conv$:

$x \in ith\ B\ i \iff (\exists f \in B. x = f\ i)$
 ⟨*proof*⟩

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

definition $eq\ upto :: (nat \Rightarrow 'a)\ set \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow (nat \Rightarrow 'a)\ set$

where

$eq\ upto\ B\ f\ i = \{g \in B. \forall j < i. f\ j = g\ j\}$

lemma $eq\ uptoI$ [*intro*]:

$\llbracket g \in B; \bigwedge j. j < i \implies f\ j = g\ j \rrbracket \implies g \in eq\ upto\ B\ f\ i$
 ⟨*proof*⟩

lemma $eq\ uptoE$ [*elim*]:

$\llbracket g \in eq\ upto\ B\ f\ i; \llbracket g \in B; \bigwedge j. j < i \implies f\ j = g\ j \rrbracket \implies Q \rrbracket \implies Q$
 ⟨*proof*⟩

lemma $eq\ upto\ Suc$:

$\llbracket g \in eq\ upto\ B\ f\ i; g\ i = f\ i \rrbracket \implies g \in eq\ upto\ B\ f\ (Suc\ i)$
 ⟨*proof*⟩

lemma $eq\ upto\ 0$ [*simp*]:

$eq\ upto\ B\ f\ 0 = B$
 ⟨*proof*⟩

lemma $eq\ upto\ cong$ [*fundef-cong*]:

assumes $\bigwedge j. j < i \implies f\ j = g\ j$ **and** $B = C$
shows $eq\ upto\ B\ f\ i = eq\ upto\ C\ g\ i$
 ⟨*proof*⟩

1.1 Lexicographic Order on Infinite Sequences

definition $LEX\ P\ f\ g \iff (\exists i::nat. P\ (f\ i)\ (g\ i) \wedge (\forall j < i. f\ j = g\ j))$

abbreviation $LEXEQ\ P \equiv (LEX\ P)^{==}$

lemma $LEX\ imp\ not\ LEX$:

assumes $LEX\ P\ f\ g$

and [*dest*]: $\bigwedge x y z. P x y \implies P y z \implies P x z$
and [*simp*]: $\bigwedge x. \neg P x x$
shows $\neg LEX P g f$
 <proof>

lemma *LEX-cases*:
assumes *LEX* *P f g*
obtains $(eq) f = g \mid (neq) k$ **where** $\forall i < k. f i = g i$ **and** $P (f k) (g k)$
 <proof>

lemma *LEX-imp-less*:
assumes $\forall x \in A. \neg P x x$ **and** $f \in SEQ A \vee g \in SEQ A$
and *LEX* *P f g* **and** $\forall i < k. f i = g i$ **and** $f k \neq g k$
shows $P (f k) (g k)$
 <proof>

end

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

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

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

definition *min-elt* $B = (SOME x. x \in B \wedge (\forall y \in A. P y x \longrightarrow y \notin B))$

lemma *minimal*:
assumes $x \in A$ **and** $Q x$
shows $\exists y \in A. P y x \wedge Q y \wedge (\forall z \in A. P z y \longrightarrow \neg Q z)$
 <proof>

lemma *min-elt-ex*:
assumes $B \subseteq A$ **and** $B \neq \{\}$
shows $\exists x. x \in B \wedge (\forall y \in A. P y x \longrightarrow y \notin B)$
 <proof>

lemma *min-elt-mem*:
assumes $B \subseteq A$ **and** $B \neq \{\}$
shows *min-elt* $B \in B$

<proof>

lemma *min-elt-minimal*:

assumes *: $B \subseteq A$ $B \neq \{\}$

assumes $y \in A$ **and** $P y$ (*min-elt B*)

shows $y \notin B$

<proof>

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

fun *lexmin*

where

lexmin: $\text{lexmin } C \ i = \text{min-elt } (\text{ith } (\text{eq-upto } C \ (\text{lexmin } C) \ i) \ i)$

declare *lexmin* [*simp del*]

lemma *eq-upto-lexmin-non-empty*:

assumes $C \subseteq \text{SEQ } A$ **and** $C \neq \{\}$

shows $\text{eq-upto } C \ (\text{lexmin } C) \ i \neq \{\}$

<proof>

lemma *lexmin-SEQ-mem*:

assumes $C \subseteq \text{SEQ } A$ **and** $C \neq \{\}$

shows $\text{lexmin } C \in \text{SEQ } A$

<proof>

lemma *non-empty-ith*:

assumes $C \subseteq \text{SEQ } A$ **and** $C \neq \{\}$

shows $\text{ith } (\text{eq-upto } C \ (\text{lexmin } C) \ i) \ i \subseteq A$

and $\text{ith } (\text{eq-upto } C \ (\text{lexmin } C) \ i) \ i \neq \{\}$

<proof>

lemma *lexmin-minimal*:

$C \subseteq \text{SEQ } A \implies C \neq \{\} \implies y \in A \implies P y \ (\text{lexmin } C \ i) \implies y \notin \text{ith } (\text{eq-upto } C \ (\text{lexmin } C) \ i) \ i$

<proof>

lemma *lexmin-mem*:

$C \subseteq \text{SEQ } A \implies C \neq \{\} \implies \text{lexmin } C \ i \in \text{ith } (\text{eq-upto } C \ (\text{lexmin } C) \ i) \ i$

<proof>

lemma *LEX-chain-on-eq-upto-imp-ith-chain-on*:

assumes *chain-on* (*LEX P*) (*eq-upto C f i*) (*SEQ A*)

shows *chain-on P* (*ith (eq-upto C f i) i*) *A*

<proof>

end

end

3 Enumerations of Well-Ordered Sets in Increasing Order

```
theory Least-Enum
imports Main
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  $\Rightarrow$  'a where
  enum 0 = (LEAST n. P n) |
  enum (Suc i) = (LEAST n. n > enum i  $\wedge$  P n)
```

```
lemma enum-mono:
  shows enum i < enum (Suc i)
  <proof>
```

```
lemma enum-less:
  i < j  $\implies$  enum i < enum j
  <proof>
```

```
lemma enum-P:
  shows P (enum i)
  <proof>
```

```
end
```

```
locale infinitely-many2 =
  fixes P :: 'a :: wellorder  $\Rightarrow$  'a  $\Rightarrow$  bool
  and N :: 'a
  assumes infm:  $\forall i \geq N. \exists j > i. P i j$ 
begin
```

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

```
fun enumchain :: nat  $\Rightarrow$  'a where
  enumchain 0 = N |
  enumchain (Suc n) = (LEAST m. m > enumchain n  $\wedge$  P (enumchain n) m)
```

```
lemma enumchain-mono:
  shows  $N \leq$  enumchain i  $\wedge$  enumchain i < enumchain (Suc i)
  <proof>
```

```
lemma enumchain-chain:
  shows P (enumchain i) (enumchain (Suc i))
```

<proof>

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'*:

$x \leq \text{Suc } y \longleftrightarrow x = 0 \vee (\exists x'. x = \text{Suc } x' \wedge x' \leq y)$

<proof>

lemma *ex-leq-Suc*:

$(\exists i \leq \text{Suc } j. P \ i) \longleftrightarrow P \ 0 \vee (\exists i \leq j. P \ (\text{Suc } i))$

<proof>

lemma *ex-less-Suc*:

$(\exists i < \text{Suc } j. P \ i) \longleftrightarrow P \ 0 \vee (\exists i < j. P \ (\text{Suc } i))$

<proof>

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* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{bool}$

where

$\text{good } P \ f \longleftrightarrow (\exists i \ j. i < j \wedge P \ (f \ i) \ (f \ j))$

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

abbreviation $\text{bad } P \ f \equiv \neg \text{good } P \ f$

lemma *goodI*:

$\llbracket i < j; P \ (f \ i) \ (f \ j) \rrbracket \Longrightarrow \text{good } P \ f$

<proof>

lemma *goodE* [*elim*]:

good $P f \implies (\bigwedge i j. \llbracket i < j; P (f i) (f j) \rrbracket \implies Q) \implies Q$
 ⟨proof⟩

lemma *badE* [*elim*]:
bad $P f \implies ((\bigwedge i j. i < j \implies \neg P (f i) (f j)) \implies Q) \implies Q$
 ⟨proof⟩

definition *almost-full-on* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a set ⇒ bool
where

almost-full-on $P A \longleftrightarrow (\forall f \in SEQ A. \text{good } P f)$

lemma *almost-full-onI* [*Pure.intro*]:
 $(\bigwedge f. \forall i. f i \in A \implies \text{good } P f) \implies \text{almost-full-on } P A$
 ⟨proof⟩

lemma *almost-full-onD*:
fixes $f :: nat \Rightarrow 'a$ **and** $A :: 'a \text{ set}$
assumes *almost-full-on* $P A$ **and** $\bigwedge i. f i \in A$
obtains $i j$ **where** $i < j$ **and** $P (f i) (f j)$
 ⟨proof⟩

4.2 An equivalent inductive definition

inductive *af* **for** A

where

now: $(\bigwedge x y. x \in A \implies y \in A \implies P x y) \implies \text{af } A P$
 | *later*: $(\bigwedge x. x \in A \implies \text{af } A (\lambda y z. P y z \vee P x y)) \implies \text{af } A P$

lemma *af-imp-almost-full-on*:
assumes *af* $A P$
shows *almost-full-on* $P A$
 ⟨proof⟩

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* $A Q$
 ⟨proof⟩

lemma *accessible-on-imp-af*:
assumes *accessible-on* $P A x$
shows *af* $A (\lambda u v. \neg P v u \vee \neg P u x)$
 ⟨proof⟩

lemma *wfp-on-imp-af*:
assumes *wfp-on* $P A$
shows *af* $A (\lambda x y. \neg P y x)$
 ⟨proof⟩

lemma *af-leq*:

af UNIV ((\leq) :: nat \Rightarrow nat \Rightarrow bool)
<proof>

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

lemma *not-af*:

$\neg \text{af } A P \Longrightarrow (\exists x y. x \in A \wedge y \in A \wedge \neg P x y) \wedge (\exists x \in A. \neg \text{af } A (\lambda y z. P y z \vee P x y))$
<proof>

fun *F*

where

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

lemma *almost-full-on-imp-af*:

assumes *af: almost-full-on P A*
shows *af A P*
<proof>

hide-const *NOTAF F*

lemma *almost-full-on-UNIV*:

almost-full-on ($\lambda - . \text{True}$) UNIV
<proof>

lemma *almost-full-on-imp-reflp-on*:

assumes *almost-full-on P A*
shows *reflp-on A P*
<proof>

lemma *almost-full-on-subset*:

$A \subseteq B \Longrightarrow \text{almost-full-on } P B \Longrightarrow \text{almost-full-on } P A$
<proof>

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*
<proof>

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 $\forall i::\text{nat}. f i \in A$
shows $\exists \varphi::\text{nat} \Rightarrow \text{nat}. \forall i j. i < j \longrightarrow \varphi i < \varphi j \wedge P (f (\varphi i)) (f (\varphi j))$

<proof>

Almost full relations do not admit infinite antichains.

lemma *almost-full-on-imp-no-antichain-on*:

assumes *almost-full-on P A*

shows \neg *antichain-on P f A*

<proof>

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 $h \text{ ' } A \subseteq B$

shows *almost-full-on* $(\lambda x y. Q (h x) (h y)) A$ (**is almost-full-on** ?P A)

<proof>

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

lemma *almost-full-on-hom*:

fixes $h :: 'a \Rightarrow 'b$

assumes *hom*: $\bigwedge x y. \llbracket x \in A; y \in A; P x y \rrbracket \Longrightarrow Q (h x) (h y)$

and *af*: *almost-full-on P A*

shows *almost-full-on Q (h ' A)*

<proof>

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

lemma *almost-full-on-mon*:

assumes *mon*: $\bigwedge x y. \llbracket x \in A; y \in A \rrbracket \Longrightarrow P x y = Q (h x) (h y)$ *bij-betw h A B*

and *af*: *almost-full-on Q B*

shows *almost-full-on P A*

<proof>

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>

lemma *Nil-imp-good-list-emb [simp]*:

assumes $f i = []$

shows *good (list-emb P) f*

<proof>

lemma *ne-lists*:

assumes $xs \neq []$ **and** $xs \in \text{lists } A$

shows $hd xs \in A$ **and** $tl xs \in \text{lists } A$

<proof>

lemma *list-emb-eq-length-induct [consumes 2, case-names Nil Cons]*:

assumes $\text{length } xs = \text{length } ys$

and *list-emb* P xs ys
and Q xs ys
and $\bigwedge x y xs ys. [P x y; list-emb P xs ys; Q xs ys] \implies Q (x\#xs) (y\#ys)$
shows $Q xs ys$
 $\langle proof \rangle$

lemma *list-emb-eq-length-P*:
assumes $length\ xs = length\ ys$
and *list-emb* P xs ys
shows $\forall i < length\ xs. P (xs ! i) (ys ! i)$
 $\langle proof \rangle$

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
 $\langle proof \rangle$

lemma *eq-almost-full-on-finite-set*:
assumes *finite* A
shows *almost-full-on* $(=)$ A
 $\langle proof \rangle$

4.4 Further Results

lemma *af-trans-extension-imp-wf*:
assumes *subrel*: $\bigwedge x y. P x y \implies Q x y$
and *af*: *almost-full-on* P A
and *trans*: *transp-on* A Q
shows *wfp-on* (*strict* Q) A
 $\langle proof \rangle$

lemma *af-trans-imp-wf*:
assumes *almost-full-on* P A
and *transp-on* A P
shows *wfp-on* (*strict* P) A
 $\langle proof \rangle$

lemma *wf-and-no-antichain-imp-wo-extension-wf*:
assumes *wf*: *wfp-on* (*strict* P) A
and *anti*: $\neg (\exists f. antichain-on\ P\ f\ A)$
and *subrel*: $\forall x \in A. \forall y \in A. P x y \implies Q x y$
and *qo*: *qo-on* Q A
shows *wfp-on* (*strict* Q) A
 $\langle proof \rangle$

lemma *every-go-extension-wf-imp-af*:
assumes *ext*: $\forall Q. (\forall x \in A. \forall y \in A. P\ x\ y \longrightarrow Q\ x\ y) \wedge$
 $go\text{-on}\ Q\ A \longrightarrow wfp\text{-on}\ (strict\ Q)\ A$
and *go-on P A*
shows *almost-full-on P A*
 $\langle proof \rangle$

end

5 Constructing Minimal Bad Sequences

theory *Minimal-Bad-Sequences*

imports

Almost-Full

Minimal-Elements

begin

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

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

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

lemma *minimal*:
assumes $x \in A$ **and** $P\ x$
shows $\exists y \in A. size\ y \leq size\ x \wedge P\ y \wedge (\forall z \in A. size\ z < size\ y \longrightarrow \neg P\ z)$
 $\langle proof \rangle$

lemma *less-not-eq [simp]*:
 $x \in A \implies size\ x < size\ y \implies x = y \implies False$
 $\langle proof \rangle$

The set of all bad sequences over A .

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

lemma *BAD-iff [iff]*:
 $f \in BAD\ P \iff (\forall i. f\ i \in A) \wedge bad\ P\ f$
 $\langle proof \rangle$

A partial order on infinite bad sequences.

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

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

The strict part of the above order.

definition $gseq :: ((nat \Rightarrow 'a) \times (nat \Rightarrow 'a))$ set **where**

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

lemma $geseq$ -iff:

$(f, g) \in geseq \longleftrightarrow$
 $f \in SEQ\ A \wedge g \in SEQ\ A \wedge (f = g \vee (\exists i. size\ (g\ i) < size\ (f\ i) \wedge (\forall j < i. f\ j = g\ j)))$
 $\langle proof \rangle$

lemma $gseq$ -iff:

$(f, g) \in gseq \longleftrightarrow f \in SEQ\ A \wedge g \in SEQ\ A \wedge (\exists i. size\ (g\ i) < size\ (f\ i) \wedge (\forall j < i. f\ j = g\ j))$
 $\langle proof \rangle$

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 $\bigwedge i. \llbracket \forall i. f\ i \in A; \forall i. g\ i \in A; size\ (g\ i) < size\ (f\ i); \forall j < i. f\ j = g\ j \rrbracket \Longrightarrow Q$
shows Q
 $\langle proof \rangle$

lemma $gseqE$:

assumes $(f, g) \in gseq$
and $\bigwedge i. \llbracket \forall i. f\ i \in A; \forall i. g\ i \in A; size\ (g\ i) < size\ (f\ i); \forall j < i. f\ j = g\ j \rrbracket \Longrightarrow Q$
shows Q
 $\langle proof \rangle$

sublocale min -elt-size?: *minimal-element measure-on size UNIV A*

rewrites *measure-on size UNIV* $\equiv \lambda x\ y. size\ x < size\ y$
 $\langle proof \rangle$

context

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

begin

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

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

lemma eq -upto-BAD-mem:

assumes $f \in eq$ -upto $(BAD\ P)\ g\ i$
shows $f\ j \in A$
 $\langle proof \rangle$

Assume that there is some infinite bad sequence h .

context

fixes $h :: nat \Rightarrow 'a$
assumes $BAD\text{-}ex: h \in BAD\ P$
begin

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

lemma $eq\text{-}upto\text{-}BAD\text{-}non\text{-}empty$:
 $eq\text{-}upto (BAD\ P) lb\ i \neq \{\}$
 $\langle proof \rangle$

lemma $non\text{-}empty\text{-}ith$:
shows $ith (eq\text{-}upto (BAD\ P) lb\ i) i \subseteq A$
and $ith (eq\text{-}upto (BAD\ P) lb\ i) i \neq \{\}$
 $\langle proof \rangle$

lemmas
 $lb\text{-}minimal = min\text{-}elt\text{-}minimal [OF\ non\text{-}empty\text{-}ith, folded\ lexmin]$ **and**
 $lb\text{-}mem = min\text{-}elt\text{-}mem [OF\ non\text{-}empty\text{-}ith, folded\ lexmin]$

lb is a infinite bad sequence.

lemma $lb\text{-}BAD$:
 $lb \in BAD\ P$
 $\langle proof \rangle$

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

lemma $lb\text{-}lower\text{-}bound$:
 $\forall g. (lb, g) \in gseq \longrightarrow g \notin BAD\ P$
 $\langle proof \rangle$

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

lemma $lower\text{-}bound\text{-}ex$:
 $\exists f \in BAD\ P. \forall g. (f, g) \in gseq \longrightarrow g \notin BAD\ P$
 $\langle proof \rangle$

lemma $gseq\text{-}conv$:
 $(f, g) \in gseq \longleftrightarrow f \neq g \wedge (f, g) \in gseq$
 $\langle proof \rangle$

There is a minimal bad sequence.

lemma mbs :
 $\exists f \in BAD\ P. \forall g. (f, g) \in gseq \longrightarrow good\ P\ g$
 $\langle proof \rangle$

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
<proof>
```

```
lemma po-on-strict-suffix:
  po-on strict-suffix A
<proof>
```

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>
```

```
lemma LEX-trans:
  assumes transp-on A P and f ∈ SEQ A and g ∈ SEQ A and h ∈ SEQ A
  and LEX P f g and LEX P g h
  shows LEX P f h
<proof>
```

```
lemma qo-on-LEXEQ:
  transp-on A P ⇒ qo-on (LEXEQ P) (SEQ A)
<proof>
```

```
context minimal-element
begin
```

```
lemma glb-LEX-lexmin:
  assumes chain-on (LEX P) C (SEQ A) and C ≠ {}
  shows glb (LEX P) C (lexmin C)
<proof>
```

```
lemma dc-on-LEXEQ:
  dc-on (LEXEQ P) (SEQ A)
```

<proof>

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 $Q A \implies Q f \implies f \in A \implies (\exists n. (\forall g \in A. (\forall i < n. g i = f i) \longrightarrow Q g))$

<proof>

lemma *pt-open-on-good*:

pt-open-on (*good* Q) (*SEQ* A)

<proof>

context *minimal-element*

begin

lemma *pt-open-on-imp-open-on-LEXEQ*:

assumes *pt-open-on* Q (*SEQ* A)

shows *open-on* (*LEXEQ* P) Q (*SEQ* A)

<proof>

lemma *open-on-good*:

open-on (*LEXEQ* P) (*good* Q) (*SEQ* A)

<proof>

end

lemma *open-on-LEXEQ-imp-pt-open-on-counterexample*:

fixes $a b :: 'a$

defines $A \equiv \{a, b\}$ **and** $P \equiv (\lambda x y. False)$ **and** $Q \equiv (\lambda f. \forall i. f i = b)$

assumes [*simp*]: $a \neq b$

shows *minimal-element* $P A$ **and** *open-on* (*LEXEQ* P) Q (*SEQ* A)

and \neg *pt-open-on* Q (*SEQ* A)

<proof>

lemma *higman*:

assumes *almost-full-on* $P A$

shows *almost-full-on* (*list-emb* P) (*lists* A)

<proof>

end

7 Almost-Full Relations

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

```
lemma (in mbs) mbs':
  assumes  $\neg$  almost-full-on P A
  shows  $\exists m \in \text{BAD } P. \forall g. (m, g) \in \text{gseq} \longrightarrow \text{good } P \ g$ 
  <proof>
```

7.1 Adding a Bottom Element to a Set

```
definition with-bot :: 'a set  $\Rightarrow$  'a option set (<-⊥> [1000] 1000)
where
```

```
   $A_{\perp} = \{\text{None}\} \cup \text{Some } 'A$ 
```

```
lemma with-bot-iff [iff]:
   $\text{Some } x \in A_{\perp} \longleftrightarrow x \in A$ 
  <proof>
```

```
lemma NoneI [simp, intro]:
   $\text{None} \in A_{\perp}$ 
  <proof>
```

```
lemma not-None-the-mem [simp]:
   $x \neq \text{None} \Longrightarrow \text{the } x \in A \longleftrightarrow x \in A_{\perp}$ 
  <proof>
```

```
lemma with-bot-cases:
   $u \in A_{\perp} \Longrightarrow (\bigwedge x. x \in A \Longrightarrow u = \text{Some } x \Longrightarrow P) \Longrightarrow (u = \text{None} \Longrightarrow P) \Longrightarrow P$ 
  <proof>
```

```
lemma with-bot-empty-conv [iff]:
   $A_{\perp} = \{\text{None}\} \longleftrightarrow A = \{\}$ 
  <proof>
```

```
lemma with-bot-UNIV [simp]:
   $\text{UNIV}_{\perp} = \text{UNIV}$ 
  <proof>
```

7.2 Adding a Bottom Element to an Almost-Full Set

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

lemma *None-imp-good-option-le* [simp]:

assumes $f\ i = \text{None}$
shows $\text{good } (\text{option-le } P)\ f$
 $\langle \text{proof} \rangle$

lemma *almost-full-on-with-bot*:

assumes $\text{almost-full-on } P\ A$
shows $\text{almost-full-on } (\text{option-le } P)\ A_{\perp}$ (**is** $\text{almost-full-on } ?P\ ?A$)
 $\langle \text{proof} \rangle$

7.3 Disjoint Union of Almost-Full Sets

fun

$\text{sum-le} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('c \Rightarrow 'd \Rightarrow \text{bool}) \Rightarrow 'a + 'b \Rightarrow 'c + 'd \Rightarrow \text{bool}$

where

$\text{sum-le } P\ Q\ (\text{Inl } x)\ (\text{Inl } y) = P\ x\ y \mid$
 $\text{sum-le } P\ Q\ (\text{Inr } x)\ (\text{Inr } y) = Q\ x\ y \mid$
 $\text{sum-le } P\ Q\ x\ y = \text{False}$

lemma *not-sum-le-cases*:

assumes $\neg \text{sum-le } P\ Q\ a\ b$
and $\bigwedge x\ y. \llbracket a = \text{Inl } x; b = \text{Inl } y; \neg P\ x\ y \rrbracket \Longrightarrow \text{thesis}$
and $\bigwedge x\ y. \llbracket a = \text{Inr } x; b = \text{Inr } y; \neg Q\ x\ y \rrbracket \Longrightarrow \text{thesis}$
and $\bigwedge x\ y. \llbracket a = \text{Inl } x; b = \text{Inr } y \rrbracket \Longrightarrow \text{thesis}$
and $\bigwedge x\ y. \llbracket a = \text{Inr } x; b = \text{Inl } y \rrbracket \Longrightarrow \text{thesis}$
shows thesis
 $\langle \text{proof} \rangle$

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

lemma *almost-full-on-Plus*:

assumes $\text{almost-full-on } P\ A$ **and** $\text{almost-full-on } Q\ B$
shows $\text{almost-full-on } (\text{sum-le } P\ Q)\ (A\ <+>\ B)$ (**is** $\text{almost-full-on } ?P\ ?A$)
 $\langle \text{proof} \rangle$

7.4 Dickson's Lemma for Almost-Full Relations

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

definition

$\text{prod-le} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('c \Rightarrow 'd \Rightarrow \text{bool}) \Rightarrow 'a \times 'b \Rightarrow 'c \times 'd \Rightarrow \text{bool}$

where

$\text{prod-le } P1\ P2 = (\lambda(p1, p2)\ (q1, q2). P1\ p1\ q1 \wedge P2\ p2\ q2)$

lemma *prod-le-True* [simp]:

$\text{prod-le } P\ (\lambda\ -.\ \text{True})\ a\ b = P\ (\text{fst } a)\ (\text{fst } b)$
 $\langle \text{proof} \rangle$

lemma *almost-full-on-Sigma*:

assumes $\text{almost-full-on } P1\ A1$ **and** $\text{almost-full-on } P2\ A2$
shows $\text{almost-full-on } (\text{prod-le } P1\ P2)\ (A1 \times A2)$ (**is** $\text{almost-full-on } ?P\ ?A$)

<proof>

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>

7.6 Natural Numbers

lemma *almost-full-on-UNIV-nat*:

almost-full-on (\leq) (UNIV :: nat set)

<proof>

end

8 Well-Quasi-Orders

theory *Well-Quasi-Orders*

imports *Almost-Full-Relations*

begin

8.1 Basic Definitions

definition *wqo-on* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**

wqo-on P A \iff transp-on A P \wedge almost-full-on P A

lemma *wqo-on-UNIV*:

wqo-on (λ -. True) UNIV

<proof>

lemma *wqo-onI* [*Pure.intro*]:

$\llbracket \text{transp-on } A \ P; \text{ almost-full-on } P \ A \rrbracket \implies \text{wqo-on } P \ A$

<proof>

lemma *wqo-on-imp-reflp-on*:

wqo-on P A \implies reflp-on A P

<proof>

lemma *wqo-on-imp-transp-on*:

wqo-on P A \implies transp-on A P

<proof>

lemma *wqo-on-imp-almost-full-on*:

wqo-on P A \implies almost-full-on P A

<proof>

lemma *wqo-on-imp-qo-on*:

$wqo\text{-on } P A \implies qo\text{-on } P A$
 ⟨proof⟩

lemma *wqo-on-imp-good*:
 $wqo\text{-on } P A \implies \forall i. f i \in A \implies good P f$
 ⟨proof⟩

lemma *wqo-on-subset*:
 $A \subseteq B \implies wqo\text{-on } P B \implies wqo\text{-on } P A$
 ⟨proof⟩

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 $qo\text{-on } P A$
shows $wqo\text{-on } P A \longleftrightarrow almost\text{-full-on } P A$
 ⟨proof⟩

lemma *wqo-wf-and-no-antichain-conv*:
assumes $qo\text{-on } P A$
shows $wqo\text{-on } P A \longleftrightarrow wfp\text{-on } (strict P) A \wedge \neg (\exists f. antichain\text{-on } P f A)$
 ⟨proof⟩

lemma *wqo-extensions-wf-conv*:
assumes $qo\text{-on } P A$
shows $wqo\text{-on } P A \longleftrightarrow (\forall Q. (\forall x \in A. \forall y \in A. P x y \longrightarrow Q x y) \wedge qo\text{-on } Q A \longrightarrow wfp\text{-on } (strict Q) A)$
 ⟨proof⟩

lemma *wqo-on-imp-wfp-on*:
 $wqo\text{-on } P A \implies wfp\text{-on } (strict P) A$
 ⟨proof⟩

The homomorphic image of a wqo set is wqo.

lemma *wqo-on-hom*:
assumes $transp\text{-on } (h \text{ ` } A) Q$
and $\forall x \in A. \forall y \in A. P x y \longrightarrow Q (h x) (h y)$
and $wqo\text{-on } P A$
shows $wqo\text{-on } Q (h \text{ ` } A)$
 ⟨proof⟩

The monomorphic preimage of a wqo set is wqo.

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

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*)
 \langle *proof* \rangle

lemma *wqo-on-UNIV-class-wqo* [*intro!*]:
wqo-on *P* *UNIV* \implies *class.wqo* *P* (*strict* *P*)
 \langle *proof* \rangle

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*)
 \langle *proof* \rangle

The strict part of a wqo is well-founded.

lemma (**in** *wqo*) *wfP* ($<$)
 \langle *proof* \rangle

lemma *wqo-on-with-bot*:
assumes *wqo-on* *P* *A*
shows *wqo-on* (*option-le* *P*) *A* \perp (**is** *wqo-on* ?*P* ?*A*)
 \langle *proof* \rangle

lemma *wqo-on-option-UNIV* [*intro*]:
wqo-on *P* *UNIV* \implies *wqo-on* (*option-le* *P*) *UNIV*
 \langle *proof* \rangle

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*)
 \langle *proof* \rangle

lemma *wqo-on-sum-UNIV* [*intro*]:
wqo-on *P* *UNIV* \implies *wqo-on* *Q* *UNIV* \implies *wqo-on* (*sum-le* *P* *Q*) *UNIV*
 \langle *proof* \rangle

8.4 Dickson's Lemma

lemma *wqo-on-Sigma*:
 fixes $A1 :: 'a \text{ set}$ and $A2 :: 'b \text{ set}$
 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*⟩

lemmas *dickson = wqo-on-Sigma*

lemma *wqo-on-prod-UNIV [intro]*:
 $wqo-on P UNIV \implies wqo-on Q UNIV \implies wqo-on (prod-le P Q) UNIV$
 ⟨*proof*⟩

8.5 Higman's Lemma

lemma *transp-on-list-emb*:
 assumes *transp-on A P*
 shows *transp-on (lists A) (list-emb P)*
 ⟨*proof*⟩

lemma *wqo-on-lists*:
 assumes *wqo-on P A* shows *wqo-on (list-emb P) (lists A)*
 ⟨*proof*⟩

lemmas *higman = wqo-on-lists*

lemma *wqo-on-list-UNIV [intro]*:
 $wqo-on P UNIV \implies wqo-on (list-emb P) UNIV$
 ⟨*proof*⟩

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*
 ⟨*proof*⟩

lemma *finite-eq-wqo-on*:
 assumes *finite A*
 shows *wqo-on (=) A*
 ⟨*proof*⟩

lemma *wqo-on-lists-over-finite-sets*:
 $wqo-on (list-emb (=)) (UNIV::('a::finite) \text{ list set})$
 ⟨*proof*⟩

lemma *wqo-on-map*:
 fixes P and Q and h
 defines $P' \equiv \lambda x y. P x y \wedge Q (h x) (h y)$
 assumes *wqo-on P A*

```

    and wqo-on Q B
    and subset: h ' A ⊆ B
  shows wqo-on P' A
  ⟨proof⟩

lemma wqo-on-UNIV-nat:
  wqo-on (≤) (UNIV :: nat set)
  ⟨proof⟩

end

```

9 Kruskal's Tree Theorem

```

theory Kruskal
imports Well-Quasi-Orders
begin

```

```

locale kruskal-tree =
  fixes F :: ('b × nat) set
  and mk :: 'b ⇒ 'a list ⇒ ('a::size)
  and root :: 'a ⇒ 'b × nat
  and args :: 'a ⇒ 'a list
  and trees :: 'a set
  assumes size-arg: t ∈ trees ⇒ s ∈ set (args t) ⇒ size s < size t
  and root-mk: (f, length ts) ∈ F ⇒ root (mk f ts) = (f, length ts)
  and args-mk: (f, length ts) ∈ F ⇒ args (mk f ts) = ts
  and mk-root-args: t ∈ trees ⇒ mk (fst (root t)) (args t) = t
  and trees-root: t ∈ trees ⇒ root t ∈ F
  and trees-arity: t ∈ trees ⇒ length (args t) = snd (root t)
  and trees-args: ∧ s. t ∈ trees ⇒ s ∈ set (args t) ⇒ s ∈ trees
begin

```

```

lemma mk-inject [iff]:
  assumes (f, length ss) ∈ F and (g, length ts) ∈ F
  shows mk f ss = mk g ts ⟷ f = g ∧ ss = ts
  ⟨proof⟩

```

```

inductive emb for P
where

```

```

  arg: [(f, m) ∈ F; length ts = m; ∀ t ∈ set ts. t ∈ trees;
    t ∈ set ts; emb P s t] ⇒ emb P s (mk f ts) |
  list-emb: [(f, m) ∈ F; (g, n) ∈ F; length ss = m; length ts = n;
    ∀ s ∈ set ss. s ∈ trees; ∀ t ∈ set ts. t ∈ trees;
    P (f, m) (g, n); list-emb (emb P) ss ts] ⇒ emb P (mk f ss) (mk g ts)
monos list-emb-mono

```

```

lemma almost-full-on-trees:
  assumes almost-full-on P F
  shows almost-full-on (emb P) trees (is almost-full-on ?P ?A)

```

<proof>

inductive-cases

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

inductive-cases

list-emb-Nil2-cases: *list-emb P xs []* **and**

list-emb-Cons-cases: *list-emb P xs (y#ys)*

lemma *list-emb-trans-right*:

assumes *list-emb P xs ys* **and** *list-emb* $(\lambda y z. P y z \wedge (\forall x. P x y \longrightarrow P x z)) ys$

zs

shows *list-emb P xs zs*

<proof>

lemma *emb-trans*:

assumes *trans*: $\bigwedge f g h. f \in F \implies g \in F \implies h \in F \implies P f g \implies P g h \implies P f h$

assumes *emb P s t* **and** *emb P t u*

shows *emb P s u*

<proof>

lemma *transp-on-emb*:

assumes *transp-on F P*

shows *transp-on trees (emb P)*

<proof>

lemma *kruskal*:

assumes *wqo-on P F*

shows *wqo-on (emb P) trees*

<proof>

end

end

theory *Kruskal-Examples*

imports *Kruskal*

begin

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

fun *node*

where

node (Node f ts) = (f, length ts)

fun *succs*

where

succs (Node f ts) = ts

inductive-set *trees* **for** A

where

$f \in A \implies \forall t \in \text{set } ts. t \in \text{trees } A \implies \text{Node } f \text{ } ts \in \text{trees } A$

lemma [*simp*]:

$\text{trees } UNIV = UNIV$

<proof>

interpretation *kruskal-tree-tree*: $\text{kruskal-tree } A \times UNIV \text{ Node } node \text{ succs } \text{trees } A$
for A

<proof>

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

thm *kruskal-tree-tree.kruskal*

definition $\text{tree-emb } A \ P = \text{kruskal-tree-tree.emb } A \ (\text{prod-le } P \ (\lambda - \cdot. \text{True}))$

lemma *wqo-on-trees*:

assumes *wqo-on* $P \ A$

shows *wqo-on* $(\text{tree-emb } A \ P) \ (\text{trees } A)$

<proof>

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 $s \leq t \longleftrightarrow \text{tree-emb } UNIV \ (\leq) \ s \ t$

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

instance

<proof>

end

datatype $(f, 'v) \ \text{term} = \text{Var } 'v \mid \text{Fun } 'f \ (f, 'v) \ \text{term } \text{list}$

fun *root*

where

$\text{root } (\text{Fun } f \ ts) = (f, \text{length } ts)$

fun *args*

where

$\text{args } (\text{Fun } f \ ts) = ts$

inductive-set *gterms* **for** F

where

$(f, n) \in F \implies \text{length } ts = n \implies \forall s \in \text{set } ts. s \in \text{gterms } F \implies \text{Fun } f \ ts \in \text{gterms } F$

interpretation *kruskal-term*: $\text{kruskal-tree } F \ \text{Fun } \text{root } \text{args } \text{gterms } F$ **for** F

<proof>

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

inductive-set *terms*

where

$\forall t \in \text{set } ts. t \in \text{terms} \implies \text{Fun } f \text{ } ts \in \text{terms}$

interpretation *kruskal-variadic: kruskal-tree UNIV Fun root args terms*

<proof>

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

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

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

fun *mk*

where

$mk (v \ x) [] = V \ x \ |$

$mk (c \ n) [] = C \ n \ |$

$mk \ p \ [a, b] = Plus \ a \ b$

fun *rt*

where

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

$rt (C \ n) = (c \ n, 0) \ |$

$rt (Plus \ a \ b) = (p, 2)$

fun *ags*

where

$ags (V \ x) = [] \ |$

$ags (C \ n) = [] \ |$

$ags (Plus \ a \ b) = [a, b]$

inductive-set *exps*

where

$V \ x \in \text{exps} \ |$

$C \ n \in \text{exps} \ |$

$a \in \text{exps} \implies b \in \text{exps} \implies Plus \ a \ b \in \text{exps}$

lemma [*simp*]:

assumes $length \ ts = 2$

shows $rt (mk \ p \ ts) = (p, 2)$

<proof>

lemma [*simp*]:

assumes $length \ ts = 2$

shows $ags (mk \ p \ ts) = ts$

```

    <proof>

interpretation kruskal-exp: kruskal-tree
  {(v x, 0) | x. True} ∪ {(c n, 0) | n. True} ∪ {(p, 2)}
  mk rt ags exps
<proof>

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  $x \leq y \longleftrightarrow \text{option-le } (\leq) \ x \ y$ 
definition  $(x :: 'a \ \text{option}) < y \longleftrightarrow x \leq y \wedge \neg (y \leq x)$ 

instance
  <proof>
end

```

10.2 The Sum Type is Well-Quasi-Ordered

```

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

instance
  <proof>
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  $p \leq q \longleftrightarrow \text{prod-le } (\leq) \ (\leq) \ p \ q$ 

```

definition $(p :: 'a \times 'b) < q \longleftrightarrow p \leq q \wedge \neg (q \leq p)$

instance
 $\langle proof \rangle$
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 $xs \leq ys \longleftrightarrow list-emb (\leq) xs ys$

definition $(xs :: 'a list) < ys \longleftrightarrow xs \leq ys \wedge \neg (ys \leq xs)$

instance
 $\langle proof \rangle$
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*:

$M \in multisets A \longleftrightarrow set-mset M \subseteq A$

$\langle proof \rangle$

lemma *empty-multisets* [simp]:

$\{\#\} \in multisets F$

$\langle proof \rangle$

lemma *multisets-union* [simp]:

$M \in multisets A \Longrightarrow N \in multisets A \Longrightarrow M + N \in multisets A$

$\langle proof \rangle$

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

lemma *mulex1-empty* [iff]:
 $mulex1\ P\ M\ \{\#\} \longleftrightarrow False$
 ⟨proof⟩

lemma *mulex1-add*: $mulex1\ P\ N\ (M0 + \{\#a\#\}) \implies$
 $(\exists M. mulex1\ P\ M\ M0 \wedge N = M + \{\#a\#\}) \vee$
 $(\exists K. (\forall b. b \in\# K \longrightarrow P\ b\ a) \wedge N = M0 + K)$
 ⟨proof⟩

lemma *mulex1-self-add-right* [simp]:
 $mulex1\ P\ A\ (add-mset\ a\ A)$
 ⟨proof⟩

lemma *empty-mult1* [simp]:
 $(\{\#\}, \{\#a\#\}) \in mult1\ R$
 ⟨proof⟩

lemma *empty-mulex1* [simp]:
 $mulex1\ P\ \{\#\}\ \{\#a\#\}$
 ⟨proof⟩

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

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

lemma *mulex-on-induct* [consumes 1, case-names base step, induct pred: *mulex-on*]:
assumes $mulex-on\ P\ A\ M\ N$
and $\bigwedge M\ N. \llbracket M \in multisets\ A; N \in multisets\ A; mulex1\ P\ M\ N \rrbracket \implies Q\ M\ N$
and $\bigwedge L\ M\ N. \llbracket mulex-on\ P\ A\ L\ M; Q\ L\ M; N \in multisets\ A; mulex1\ P\ M\ N \rrbracket$
 $\implies Q\ L\ N$
shows $Q\ M\ N$
 ⟨proof⟩

lemma *mulex-on-self-add-singleton-right* [simp]:
assumes $a \in A$ **and** $M \in multisets\ A$
shows $mulex-on\ P\ A\ M\ (add-mset\ a\ M)$
 ⟨proof⟩

lemma *singleton-multisets* [iff]:
 $\{\#x\#\} \in multisets\ A \longleftrightarrow x \in A$
 ⟨proof⟩

lemma *union-multisetsD*:
assumes $M + N \in multisets\ A$
shows $M \in multisets\ A \wedge N \in multisets\ A$

<proof>

lemma *mulex-on-multisetsD* [*dest*]:

assumes *mulex-on P F M N*

shows $M \in \text{multisets } F$ **and** $N \in \text{multisets } F$

<proof>

lemma *union-multisets-iff* [*iff*]:

$M + N \in \text{multisets } A \longleftrightarrow M \in \text{multisets } A \wedge N \in \text{multisets } A$

<proof>

lemma *add-mset-multisets-iff* [*iff*]:

$\text{add-mset } a \ M \in \text{multisets } A \longleftrightarrow a \in A \wedge M \in \text{multisets } A$

<proof>

lemma *mulex-on-trans*:

$\text{mulex-on } P \ A \ L \ M \Longrightarrow \text{mulex-on } P \ A \ M \ N \Longrightarrow \text{mulex-on } P \ A \ L \ N$

<proof>

lemma *transp-on-mulex-on*:

$\text{transp-on } B \ (\text{mulex-on } P \ A)$

<proof>

lemma *mulex-on-add-right* [*simp*]:

assumes *mulex-on P A M N* **and** $a \in A$

shows $\text{mulex-on } P \ A \ M \ (\text{add-mset } a \ N)$

<proof>

lemma *empty-mulex-on* [*simp*]:

assumes $M \neq \{\#\}$ **and** $M \in \text{multisets } A$

shows $\text{mulex-on } P \ A \ \{\#\} \ M$

<proof>

lemma *mulex-on-self-add-right* [*simp*]:

assumes $M \in \text{multisets } A$ **and** $K \in \text{multisets } A$ **and** $K \neq \{\#\}$

shows $\text{mulex-on } P \ A \ M \ (M + K)$

<proof>

lemma *mult1-singleton* [*iff*]:

$(\{\#x\#\}, \{\#y\#\}) \in \text{mult1 } R \longleftrightarrow (x, y) \in R$

<proof>

lemma *mulex1-singleton* [*iff*]:

$\text{mulex1 } P \ \{\#x\#\} \ \{\#y\#\} \longleftrightarrow P \ x \ y$

<proof>

lemma *singleton-mulex-onI*:

$P \ x \ y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow \text{mulex-on } P \ A \ \{\#x\#\} \ \{\#y\#\}$

<proof>

lemma *reflcp-mulex-on-add-right* [simp]:
assumes $(\text{mulex-on } P \ A) = M \ N$ **and** $M \in \text{multisets } A$ **and** $a \in A$
shows $\text{mulex-on } P \ A \ M \ (N + \{\#a\#})$
 $\langle \text{proof} \rangle$

lemma *reflcp-mulex-on-add-right'* [simp]:
assumes $(\text{mulex-on } P \ A) = M \ N$ **and** $M \in \text{multisets } A$ **and** $a \in A$
shows $\text{mulex-on } P \ A \ M \ (\{\#a\#} + N)$
 $\langle \text{proof} \rangle$

lemma *mulex-on-union-right* [simp]:
assumes $\text{mulex-on } P \ F \ A \ B$ **and** $K \in \text{multisets } F$
shows $\text{mulex-on } P \ F \ A \ (K + B)$
 $\langle \text{proof} \rangle$

lemma *mulex-on-union-right'* [simp]:
assumes $\text{mulex-on } P \ F \ A \ B$ **and** $K \in \text{multisets } F$
shows $\text{mulex-on } P \ F \ A \ (B + K)$
 $\langle \text{proof} \rangle$

Adapted from $\text{wf } ?r \implies \forall M. M \in \text{Wellfounded.acc } (\text{mult1 } ?r)$ in *HOL-Library.Multiset*.

lemma *accessible-on-mulex1-multisets*:
assumes $\text{wf}: \text{wfp-on } P \ A$
shows $\forall M \in \text{multisets } A. \text{accessible-on } (\text{mulex1 } P) \ (\text{multisets } A) \ M$
 $\langle \text{proof} \rangle$

lemmas *wfp-on-mulex1-multisets* =
 $\text{accessible-on-mulex1-multisets } [\text{THEN } \text{accessible-on-imp-wfp-on}]$

lemmas *irreflp-on-mulex1* =
 $\text{wfp-on-mulex1-multisets } [\text{THEN } \text{wfp-on-imp-irreflp-on}]$

lemma *wfp-on-mulex-on-multisets*:
assumes $\text{wfp-on } P \ A$
shows $\text{wfp-on } (\text{mulex-on } P \ A) \ (\text{multisets } A)$
 $\langle \text{proof} \rangle$

lemmas *irreflp-on-mulex-on* =
 $\text{wfp-on-mulex-on-multisets } [\text{THEN } \text{wfp-on-imp-irreflp-on}]$

lemma *mulex1-union*:
 $\text{mulex1 } P \ M \ N \implies \text{mulex1 } P \ (K + M) \ (K + N)$
 $\langle \text{proof} \rangle$

lemma *mulex-on-union*:
assumes $\text{mulex-on } P \ A \ M \ N$ **and** $K \in \text{multisets } A$
shows $\text{mulex-on } P \ A \ (K + M) \ (K + N)$
 $\langle \text{proof} \rangle$

lemma *mulex-on-union'*:

assumes *mulex-on P A M N* **and** $K \in \text{multisets } A$
shows *mulex-on P A (M + K) (N + K)*
(*proof*)

lemma *mulex-on-add-mset*:

assumes *mulex-on P A M N* **and** $m \in A$
shows *mulex-on P A (add-mset m M) (add-mset m N)*
(*proof*)

lemma *union-mulex-on-mono*:

mulex-on P F A C \implies *mulex-on P F B D* \implies *mulex-on P F (A + B) (C + D)*
(*proof*)

lemma *mulex-on-add-mset'*:

assumes $P m n$ **and** $m \in A$ **and** $n \in A$ **and** $M \in \text{multisets } A$
shows *mulex-on P A (add-mset m M) (add-mset n M)*
(*proof*)

lemma *mulex-on-add-mset-mono*:

assumes $P m n$ **and** $m \in A$ **and** $n \in A$ **and** *mulex-on P A M N*
shows *mulex-on P A (add-mset m M) (add-mset n N)*
(*proof*)

lemma *union-mulex-on-mono1*:

$A \in \text{multisets } F$ \implies (*mulex-on P F*)⁼⁼ $A C$ \implies *mulex-on P F B D* \implies
mulex-on P F (A + B) (C + D)
(*proof*)

lemma *union-mulex-on-mono2*:

$B \in \text{multisets } F$ \implies *mulex-on P F A C* \implies (*mulex-on P F*)⁼⁼ $B D$ \implies
mulex-on P F (A + B) (C + D)
(*proof*)

lemma *mult1-mono*:

assumes $\bigwedge x y. \llbracket x \in A; y \in A; (x, y) \in R \rrbracket \implies (x, y) \in S$
and $M \in \text{multisets } A$
and $N \in \text{multisets } A$
and $(M, N) \in \text{mult1 } R$
shows $(M, N) \in \text{mult1 } S$
(*proof*)

lemma *mulex1-mono*:

assumes $\bigwedge x y. \llbracket x \in A; y \in A; P x y \rrbracket \implies Q x y$
and $M \in \text{multisets } A$
and $N \in \text{multisets } A$
and *mulex1 P M N*
shows *mulex1 Q M N*

<proof>

lemma *mulex-on-mono*:

assumes *: $\bigwedge x y. \llbracket x \in A; y \in A; P x y \rrbracket \implies Q x y$

and *mulex-on* $P A M N$

shows *mulex-on* $Q A M N$

<proof>

lemma *mult1-reflcl*:

assumes $(M, N) \in \text{mult1 } R$

shows $(M, N) \in \text{mult1 } (R^-)$

<proof>

lemma *mulex1-reflclp*:

assumes *mulex1* $P M N$

shows *mulex1* $(P^{==}) M N$

<proof>

lemma *mulex-on-reflclp*:

assumes *mulex-on* $P A M N$

shows *mulex-on* $(P^{==}) A M N$

<proof>

lemma *surj-on-multisets-mset*:

$\forall M \in \text{multisets } A. \exists xs \in \text{lists } A. M = \text{mset } xs$

<proof>

lemma *image-mset-lists* [*simp*]:

mset ‘ *lists* $A = \text{multisets } A$

<proof>

lemma *multisets-UNIV* [*simp*]: *multisets* $UNIV = UNIV$

<proof>

lemma *non-empty-multiset-induct* [*consumes 1, case-names singleton add*]:

assumes $M \neq \{\#\}$

and $\bigwedge x. P \{\#x\# \}$

and $\bigwedge x M. P M \implies P (\text{add-mset } x M)$

shows $P M$

<proof>

lemma *mulex-on-all-strict*:

assumes $X \neq \{\#\}$

assumes $X \in \text{multisets } A$ **and** $Y \in \text{multisets } A$

and *: $\forall y. y \in \# Y \longrightarrow (\exists x. x \in \# X \wedge P y x)$

shows *mulex-on* $P A Y X$

<proof>

The following lemma shows that the textbook definition (e.g., “Term Rewrit-

ing and All That”) is the same as the one used below.

lemma *diff-set-Ex-iff*:

$X \neq \{\#\} \wedge X \subseteq \# M \wedge N = (M - X) + Y \longleftrightarrow X \neq \{\#\} \wedge (\exists Z. M = Z + X \wedge N = Z + Y)$

<proof>

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

lemma *mulex-on-alt-def*:

assumes *trans*: *transp-on A P*

shows *mulex-on P A M N* $\longleftrightarrow M \in \text{multisets } A \wedge N \in \text{multisets } A \wedge (\exists X Y Z.$

$X \neq \{\#\} \wedge N = Z + X \wedge M = Z + Y \wedge (\forall y. y \in \# Y \longrightarrow (\exists x. x \in \# X \wedge P y x))$

(is *?P M N* \longleftrightarrow *?Q M N*)

<proof>

end

12 Multiset Extension Preserves Well-Quasi-Orders

theory *Wqo-Multiset*

imports

Multiset-Extension

Well-Quasi-Orders

begin

lemma *list-emb-imp-reflcp-mulex-on*:

assumes *xs* \in *lists A* **and** *ys* \in *lists A*

and *list-emb P xs ys*

shows $(\text{mulex-on } P A)^{==} (\text{mset } xs) (\text{mset } ys)$

<proof>

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* $(\text{mulex-on } P A)^{==} (\text{multisets } A)$

<proof>

lemma *wqo-on-multisets*:

assumes *wqo-on P A*

shows *wqo-on* $(\text{mulex-on } P A)^{==} (\text{multisets } A)$

<proof>

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.