A Partition Theorem for the Ordinal ω^ω

Lawrence C. Paulson

December 14, 2021

Abstract

The theory of partition relations concerns generalisations of Ramsey's theorem. For any ordinal α, write $\alpha \to (\alpha, m)^2$ if for each function f from unordered pairs of elements of α into $\{0, 1\}$, either there is a subset $X \subseteq \alpha$ order-isomorphic to α such that $f\{x, y\} = 0$ for all $\{x, y\} \subseteq X$, or there is an m element set $Y \subseteq \alpha$ such that $f\{x, y\} = 1$ for all $\{x, y\} \subseteq Y$. (In both cases, with $\{x, y\}$ we require $x \neq y$.) In particular, the infinite Ramsey theorem can be written in this notation as $\omega \to (\omega, \omega)^2$, or if we restrict m to the positive integers as above, then $\omega \to (\omega, m)^2$ for all m [3].

This entry formalises Larson's proof of $\omega^\omega \to (\omega^\omega, m)^2$ along with a similar proof of a result due to Specker: $\omega^2 \to (\omega^2, m)^2$. Also proved is a necessary result by Erdős and Milner [1, 2]: $\omega^{1+\alpha\cdot n} \to (\omega^{1+\alpha}, 2^n)^2$.

These examples demonstrate the use of Isabelle/HOL to formalise advanced results that combine ZF set theory with basic concepts like lists and natural numbers.

Contents

1 Library additions

theory *Library-Additions*
imports *ZFC-in-HOL.Ordinal-Exp HOL−Library.Ramsey Nash-Williams.Nash-Williams*

begin

lemma *finite-enumerate-Diff-singleton*:
fixes S :: *'a::wellorder set*
assumes *finite* S **and** i: $i < card\ S$ *enumerate* $S\ i < x$
shows *enumerate* $(S - \{x\})\ i$ = *enumerate* $S\ i$
⟨*proof*⟩

lemma *hd-lex*: ⟦*hd ms* < *hd ns*; *length ms* = *length ns*; *ns* ≠ []⟧ ⟹ (*ms*, *ns*) ∈ *lex less-than*
⟨*proof*⟩

lemma *sorted-hd-le*:
assumes *sorted xs x* ∈ *list.set xs*
shows *hd xs* ≤ *x*
⟨*proof*⟩

lemma *sorted-le-last*:

assumes *sorted xs x* ∈ *list.set xs*
shows *x* ≤ *last xs*
⟨*proof*⟩

lemma *hd-list-of*:
assumes *finite A A* ≠ {}
shows *hd* (*sorted-list-of-set A*) = *Min A*
⟨*proof*⟩

lemma *sorted-hd-le-last*:
assumes *sorted xs xs* ≠ []
shows *hd xs* ≤ *last xs*
⟨*proof*⟩

lemma *sorted-list-of-set-set-of* [*simp*]: *strict-sorted l* ⟹ *sorted-list-of-set* (*list.set l*) = *l*
⟨*proof*⟩

lemma *range-strict-mono-ext*:
fixes *f*::*nat* ⇒ *'a*::*linorder*
assumes *eq*: *range f* = *range g*
and *sm*: *strict-mono f strict-mono g*
shows *f* = *g*
⟨*proof*⟩

1.1 Other material

definition *strict-mono-sets* :: [*'a*::*order set*, *'a*::*order* ⇒ *'b*::*order set*] ⇒ *bool* **where**
strict-mono-sets A f ≡ ∀ *x*∈*A*. ∀ *y*∈*A*. *x* < *y* ⟶ *less-sets* (*f x*) (*f y*)

lemma *strict-mono-setsD*:
assumes *strict-mono-sets A f x* < *y x* ∈ *A y* ∈ *A*
shows *less-sets* (*f x*) (*f y*)
⟨*proof*⟩

lemma *strict-mono-on-o*: ⟦*strict-mono-on r A*; *strict-mono-on s B*; *s* ‘ *B* ⊆ *A*⟧ ⟹ *strict-mono-on* (*r* ∘ *s*) *B*
⟨*proof*⟩

lemma *strict-mono-sets-imp-disjoint*:
fixes *A* :: *'a*::*linorder set*
assumes *strict-mono-sets A f*
shows *pairwise* (λ*x y*. *disjnt* (*f x*) (*f y*)) *A*
⟨*proof*⟩

lemma *strict-mono-sets-subset*:
assumes *strict-mono-sets B f A* ⊆ *B*
shows *strict-mono-sets A f*
⟨*proof*⟩

lemma *strict-mono-less-sets-Min*:
assumes *strict-mono-sets I f finite I I* $\neq \{\}$
shows *less-sets* (*f* (*Min I*)) ($\bigcup$ (*f* ' (*I* − {*Min I*})))
⟨*proof*⟩

lemma *pair-less-iff1* [*simp*]: ((*x*,*y*), (*x*,*z*)) ∈ *pair-less* ⟷ *y*<*z*
⟨*proof*⟩

lemma *infinite-finite-Inter*:
assumes *finite* $\mathcal{A}$ $\mathcal{A}\neq\{\}$ $\bigwedge A.\ A \in \mathcal{A} \Longrightarrow$ *infinite A*
and $\bigwedge A\ B.\ [\![A \in \mathcal{A};\ B \in \mathcal{A}]\!] \Longrightarrow A \cap B \in \mathcal{A}$
shows *infinite* ($\bigcap \mathcal{A}$)
⟨*proof*⟩

lemma *atLeast-less-sets*: [*less-sets A* {*x*}; *B* ⊆ {*x*..}] ⟹ *less-sets A B*
⟨*proof*⟩

1.2 The list-of function

lemma *sorted-list-of-set-insert-remove-cons*:
assumes *finite A less-sets* {*a*} *A*
shows *sorted-list-of-set* (*insert a A*) = *a* # *sorted-list-of-set A*
⟨*proof*⟩

lemma *sorted-list-of-set-Un*:
assumes *AB*: *less-sets A B* **and** *fin*: *finite A finite B*
shows *sorted-list-of-set* (*A* ∪ *B*) = *sorted-list-of-set A* @ *sorted-list-of-set B*
⟨*proof*⟩

lemma *sorted-list-of-set-UN-lessThan*:
fixes *k*::*nat*
assumes *sm*: *strict-mono-sets* {..<*k*} *A* **and** $\bigwedge i.\ i < k \Longrightarrow$ *finite* (*A i*)
shows *sorted-list-of-set* ($\bigcup i$<*k*. *A i*) = *concat* (*map* (*sorted-list-of-set* ∘ *A*) (*sorted-list-of-set* {..<*k*}))
⟨*proof*⟩

lemma *sorted-list-of-set-UN-atMost*:
fixes *k*::*nat*
assumes *strict-mono-sets* {..*k*} *A* **and** $\bigwedge i.\ i \leq k \Longrightarrow$ *finite* (*A i*)
shows *sorted-list-of-set* ($\bigcup i \leq k$. *A i*) = *concat* (*map* (*sorted-list-of-set* ∘ *A*) (*sorted-list-of-set* {..*k*}))
⟨*proof*⟩

1.3 Monotonic enumeration of a countably infinite set

abbreviation *enum* ≡ *enumerate*

Could be generalised to infinite countable sets of any type

lemma *nat-infinite-iff*:

fixes N :: *nat set*
shows *infinite* $N \longleftrightarrow (\exists f{::}nat \Rightarrow nat.\ N = range\ f \wedge strict\text{-}mono\ f)$
⟨*proof*⟩

lemma *enum-works*:
fixes N :: *nat set*
assumes *infinite* N
shows $N = range\ (enum\ N) \wedge strict\text{-}mono\ (enum\ N)$
⟨*proof*⟩

lemma *range-enum*: $range\ (enum\ N) = N$ **and** *strict-mono-enum*: *strict-mono* (*enum* N)
if *infinite* N **for** N :: *nat set*
⟨*proof*⟩

lemma *enum-0-eq-Inf*:
fixes N :: *nat set*
assumes *infinite* N
shows $enum\ N\ 0 = Inf\ N$
⟨*proof*⟩

lemma *enum-works-finite*:
fixes N :: *nat set*
assumes *finite* N
shows $N = enum\ N\ `\ \{..<card\ N\} \wedge strict\text{-}mono\text{-}on\ (enum\ N)\ \{..<card\ N\}$
⟨*proof*⟩

lemma *enum-obtain-index-finite*:
fixes N :: *nat set*
assumes $x \in N$ *finite* N
obtains i **where** $i < card\ N\ x = enum\ N\ i$
⟨*proof*⟩

lemma *enum-0-eq-Inf-finite*:
fixes N :: *nat set*
assumes *finite* $N\ N \neq \{\}$
shows $enum\ N\ 0 = Inf\ N$
⟨*proof*⟩

lemma *greaterThan-less-enum*:
fixes N :: *nat set*
assumes $N \subseteq \{x<..\}$ *infinite* N
shows $x < enum\ N\ i$
⟨*proof*⟩

lemma *atLeast-le-enum*:
fixes N :: *nat set*
assumes $N \subseteq \{x..\}$ *infinite* N
shows $x \leq enum\ N\ i$

⟨*proof*⟩

lemma *less-sets-empty1* [*simp*]: *less-sets* {} *A* **and** *less-sets-empty2* [*simp*]: *less-sets A* {}
⟨*proof*⟩

lemma *less-sets-singleton1* [*simp*]: *less-sets* {*a*} *A* ⟷ (∀ *x*∈*A*. *a* < *x*)
and *less-sets-singleton2* [*simp*]: *less-sets A* {*a*} ⟷ (∀ *x*∈*A*. *x* < *a*)
⟨*proof*⟩

lemma *less-sets-atMost* [*simp*]: *less-sets* {..*a*} *A* ⟷ (∀ *x*∈*A*. *a* < *x*)
and *less-sets-alLeast* [*simp*]: *less-sets A* {*a*..} ⟷ (∀ *x*∈*A*. *x* < *a*)
⟨*proof*⟩

lemma *less-sets-imp-strict-mono-sets*:
assumes ⋀*i*. *less-sets* (*A i*) (*A* (*Suc i*)) ⋀*i*. *i*>*0* ⟹ *A i* ≠ {}
shows *strict-mono-sets UNIV A*
⟨*proof*⟩

lemma *less-sets-Suc-Max*:
assumes *finite A*
shows *less-sets A* {*Suc* (*Max A*)..}
⟨*proof*⟩

lemma *infinite-nat-greaterThan*:
fixes *m*::*nat*
assumes *infinite N*
shows *infinite* (*N* ∩ {*m*<..})
⟨*proof*⟩

end

2 Ordinal Partitions

Material from Jean A. Larson, A short proof of a partition theorem for the ordinal ω^ω. *Annals of Mathematical Logic*, 6:129–145, 1973. Also from "Partition Relations" by A. Hajnal and J. A. Larson, in *Handbook of Set Theory*, edited by Matthew Foreman and Akihiro Kanamori (Springer, 2010).

theory *Partitions*
imports *Library-Additions ZFC-in-HOL.ZFC-Typeclasses ZFC-in-HOL.Cantor-NF*

begin

abbreviation *tp* :: *V set* ⇒ *V*
where *tp A* ≡ *ordertype A VWF*

2.1 Ordinal Partitions: Definitions

definition *partn-lst* :: $[('a \times 'a)\ set,\ 'a\ set,\ V\ list,\ nat] \Rightarrow bool$
  where *partn-lst r B α n* $\equiv \forall f \in [B]^n \rightarrow \{..<length\ \alpha\}.$
$\exists i < length\ \alpha.\ \exists H.\ H \subseteq B \wedge ordertype\ H\ r = (\alpha!i) \wedge f\ `\ (nsets\ H\ n) \subseteq \{i\}$

abbreviation *partn-lst-VWF* :: $V \Rightarrow V\ list \Rightarrow nat \Rightarrow bool$
  where *partn-lst-VWF β* $\equiv$ *partn-lst VWF* (*elts β*)

lemma *partn-lst-E*:
  assumes *partn-lst r B α n f* $\in$ *nsets B n* $\rightarrow \{..<l\}$ *l* = *length α*
  obtains *i H* **where** *i* < *l H* $\subseteq$ *B*
ordertype H r = *α*!*i f* ` (*nsets H n*) $\subseteq \{i\}$
  ⟨*proof*⟩

lemma *partn-lst-VWF-nontriv*:
  assumes *partn-lst-VWF β α n l* = *length α Ord β l* > *0*
  obtains *i* **where** *i* < *l α*!*i* $\leq$ *β*
⟨*proof*⟩

lemma *partn-lst-triv0*:
  assumes *α*!*i* = *0 i* < *length α n* $\neq$ *0*
  shows *partn-lst r B α n*
  ⟨*proof*⟩

lemma *partn-lst-triv1*:
  assumes *α*!*i* $\leq$ *1 i* < *length α n* > *1 B* $\neq \{\}$ *wf r*
  shows *partn-lst r B α n*
  ⟨*proof*⟩

lemma *partn-lst-two-swap*:
  assumes *partn-lst r B* [*x*,*y*] *n* **shows** *partn-lst r B* [*y*,*x*] *n*
⟨*proof*⟩

lemma *partn-lst-greater-resource*:
  assumes *M*: *partn-lst r B α n* **and** *B* $\subseteq$ *C*
  shows *partn-lst r C α n*
⟨*proof*⟩

lemma *partn-lst-less*:
  assumes *M*: *partn-lst r B α n* **and** *eq*: *length α′* = *length α* **and** *List.set α′* $\subseteq$ *ON*
    and *le*: $\bigwedge i.\ i < length\ \alpha \Longrightarrow \alpha'!i \leq \alpha!i$
    and *r*: *wf r trans r total-on B r* **and** *small B*
  shows *partn-lst r B α′ n*
⟨*proof*⟩

Holds because no n-sets exist!

lemma *partn-lst-VWF-degenerate*:
assumes $k < n$
shows *partn-lst-VWF* ω (*ord-of-nat* k # αs) n
⟨*proof*⟩

lemma *partn-lst-VWF-ω-2*:
assumes *Ord* α
shows *partn-lst-VWF* $(\omega \uparrow (1{+}\alpha))$ [*2*, $\omega \uparrow (1{+}\alpha)$] *2* (**is** *partn-lst-VWF ?β - -*)
⟨*proof*⟩

2.2 Relating partition properties on *VWF* to the general case

Two very similar proofs here!

lemma *partn-lst-imp-partn-lst-VWF-eq*:
assumes *part*: *partn-lst r U* α *n* **and** β: *ordertype U r* = β *small U*
and *r*: *wf r trans r total-on U r*
shows *partn-lst-VWF* β α *n*
⟨*proof*⟩

lemma *partn-lst-imp-partn-lst-VWF*:
assumes *part*: *partn-lst r U* α *n* **and** β: *ordertype U r* ≤ β *small U*
and *r*: *wf r trans r total-on U r*
shows *partn-lst-VWF* β α *n*
⟨*proof*⟩

lemma *partn-lst-VWF-imp-partn-lst-eq*:
assumes *part*: *partn-lst-VWF* β α *n* **and** β: *ordertype U r* = β *small U*
and *r*: *wf r trans r total-on U r*
shows *partn-lst r U* α *n*
⟨*proof*⟩

corollary *partn-lst-VWF-imp-partn-lst*:
assumes *partn-lst-VWF* β α *n* **and** β: *ordertype U r* ≥ β *small U*
wf r trans r total-on U r
shows *partn-lst r U* α *n*
⟨*proof*⟩

2.3 Simple consequences of the definitions

A restatement of the infinite Ramsey theorem using partition notation

lemma *Ramsey-partn*: *partn-lst-VWF* ω [ω,ω] *2*
⟨*proof*⟩

This is the counterexample sketched in Hajnal and Larson, section 9.1.

proposition *omega-basic-counterexample*:
assumes *Ord* α
shows ¬ *partn-lst-VWF* α [*succ* (*vcard* α), ω] *2*
⟨*proof*⟩

2.4 Specker's theorem

definition *form-split* :: [*nat*,*nat*,*nat*,*nat*,*nat*] ⇒ *bool* **where**
form-split a b c d i ≡ *a* ≤ *c* ∧ (*i*=*0* ∧ *a*<*b* ∧ *b*<*c* ∧ *c*<*d* ∨
i=*1* ∧ *a*<*c* ∧ *c*<*b* ∧ *b*<*d* ∨
i=*2* ∧ *a*<*c* ∧ *c*<*d* ∧ *d*<*b* ∨
i=*3* ∧ *a*=*c* ∧ *b*≠*d*)

definition *form* :: [(*nat*∗*nat*)*set*, *nat*] ⇒ *bool* **where**
form u i ≡ ∃ *a b c d*. *u* = {(*a*,*b*),(*c*,*d*)} ∧ *form-split a b c d i*

definition *scheme* :: [(*nat*∗*nat*)*set*] ⇒ *nat set* **where**
scheme u ≡ *fst* ‘ *u* ∪ *snd* ‘ *u*

definition *UU* :: (*nat*∗*nat*) *set*
where *UU* ≡ {(*a*,*b*). *a* < *b*}

lemma *ordertype-UNIV-ω2*: *ordertype UNIV pair-less* = *ω*↑*2*
⟨*proof*⟩

lemma *ordertype-UU-ge-ω2*: *ordertype UNIV pair-less* ≤ *ordertype UU pair-less*
⟨*proof*⟩

lemma *ordertype-UU-ω2*: *ordertype UU pair-less* = *ω*↑*2*
⟨*proof*⟩

Lemma 2.3 of Jean A. Larson, A short proof of a partition theorem for the ordinal ω^ω. *Annals of Mathematical Logic*, 6:129–145, 1973.

lemma *lemma-2-3*:
fixes *f* :: (*nat* × *nat*) *set* ⇒ *nat*
assumes *f* ∈ [*UU*]^2 → {..<*Suc* (*Suc 0*)}
obtains *N js* **where** *infinite N* **and** ⋀*k u*. ⟦*k* < *4*; *u* ∈ [*UU*]^2; *form u k*; *scheme u* ⊆ *N*⟧ ⟹ *f u* = *js*!*k*
⟨*proof*⟩

Lemma 2.4 of Jean A. Larson, ibid.

lemma *lemma-2-4*:
assumes *infinite N k* < *4*
obtains *M* **where** *M* ∈ [*UU*]^m ⋀*u*. *u* ∈ [*M*]^2 ⟹ *form u k* ⋀*u*. *u* ∈ [*M*]^2 ⟹ *scheme u* ⊆ *N*
⟨*proof*⟩

Lemma 2.5 of Jean A. Larson, ibid.

lemma *lemma-2-5*:
assumes *infinite N*
obtains *X* **where** *X* ⊆ *UU ordertype X pair-less* = *ω*↑*2*
⋀*u*. *u* ∈ [*X*]^2 ⟹ (∃ *k*<*4*. *form u k*) ∧ *scheme u* ⊆ *N*
⟨*proof*⟩

Theorem 2.1 of Jean A. Larson, ibid.

lemma *Specker-aux*:
assumes $\alpha \in elts\ \omega$
shows *partn-lst pair-less UU* $[\omega\uparrow 2,\alpha]\ 2$
⟨*proof*⟩

theorem *Specker*: $\alpha \in elts\ \omega \Longrightarrow$ *partn-lst-VWF* $(\omega\uparrow 2)\ [\omega\uparrow 2,\alpha]\ 2$
⟨*proof*⟩

end
theory *Erdos-Milner*
imports *Partitions*

begin

2.5 Erds-Milner theorem

P. Erds and E. C. Milner, A Theorem in the Partition Calculus. Canadian Math. Bull. 15:4 (1972), 501-505. Corrigendum, Canadian Math. Bull. 17:2 (1974), 305.

The paper defines strong types as satisfying the criteria below. It remarks that ordinals satisfy those criteria. Here is a (too complicated) proof.

proposition *strong-ordertype-eq*:
assumes *D*: $D \subseteq elts\ \beta$ **and** *Ord* β
obtains *L* **where** $\bigcup(List.set\ L) = D\ \bigwedge X.\ X \in List.set\ L \Longrightarrow indecomposable\ (tp\ X)$
and $\bigwedge M.\ [\![M \subseteq D;\ \bigwedge X.\ X \in List.set\ L \Longrightarrow tp\ (M \cap X) \geq tp\ X]\!] \Longrightarrow tp\ M = tp\ D$
⟨*proof*⟩

The "remark" of Erds and E. C. Milner, Canad. Math. Bull. Vol. 17 (2), 1974

proposition *indecomposable-imp-Ex-less-sets*:
assumes *indec*: *indecomposable* α **and** $\alpha \geq \omega$
and *A*: $tp\ A = \alpha$ *small A* $A \subseteq ON$
and $x \in A$ **and** *A1*: $tp\ A1 = \alpha\ A1 \subseteq A$
obtains *A2* **where** $tp\ A2 = \alpha\ A2 \subseteq A1\ \{x\} \ll A2$
⟨*proof*⟩

the main theorem, from which they derive the headline result

theorem *Erdos-Milner-aux*:
assumes *part*: *partn-lst-VWF* $\alpha\ [k, \gamma]\ 2$
and *indec*: *indecomposable* α **and** $k > 1$ *Ord* γ **and** β: $\beta \in elts\ \omega 1$
shows *partn-lst-VWF* $(\alpha * \beta)$ [*ord-of-nat* $(2*k)$, *min* $\gamma\ (\omega * \beta)$] 2
⟨*proof*⟩

theorem *Erdos-Milner*:
assumes ν: $\nu \in elts\ \omega 1$

shows *partn-lst-VWF* $(\omega\uparrow(1 + \nu * n))$ $[ord\text{-}of\text{-}nat\ (2\hat{}n), \omega\uparrow(1+\nu)]$ *2*
⟨*proof*⟩

corollary *remark-3*: *partn-lst-VWF* $(\omega\uparrow(Suc(4*k)))$ $[4, \omega\uparrow(Suc(2*k))]$ *2*
⟨*proof*⟩

Theorem 3.2 of Jean A. Larson, ibid.

corollary *Theorem-3-2*:
fixes *k n*::*nat*
shows *partn-lst-VWF* $(\omega\uparrow(n*k))$ $[\omega\uparrow n, ord\text{-}of\text{-}nat\ k]$ *2*
⟨*proof*⟩

end

3 An ordinal partition theorem by Jean A. Larson

Jean A. Larson, A short proof of a partition theorem for the ordinal ω^ω. *Annals of Mathematical Logic*, 6:129–145, 1973.

theory *Omega-Omega*
imports *HOL−Library.Product-Lexorder Erdos-Milner*

begin

abbreviation *list-of* ≡ *sorted-list-of-set*

3.1 Cantor normal form for ordinals below $\omega \uparrow \omega$

Unlike *Cantor-sum*, there is no list of ordinal exponents, which are instead taken as consecutive. We obtain an order-isomorphism between $\omega \uparrow \omega$ and increasing lists of natural numbers (ordered lexicographically).

fun *omega-sum-aux* **where**
Nil: *omega-sum-aux 0 - = 0*
| *Suc*: *omega-sum-aux* (*Suc n*) [] = *0*
| *Cons*: *omega-sum-aux* (*Suc n*) (*m*#*ms*) = $(\omega\uparrow n) * (ord\text{-}of\text{-}nat\ m)$ + *omega-sum-aux n ms*

abbreviation *omega-sum* **where** *omega-sum ms* ≡ *omega-sum-aux* (*length ms*) *ms*

A normal expansion has no leading zeroes

inductive *normal*:: *nat list* ⇒ *bool* **where**
normal-Nil[*iff*]: *normal* []
| *normal-Cons*: $m > 0 \Longrightarrow$ *normal* (*m*#*ms*)

inductive-simps *normal-Cons-iff* [*simp*]: *normal* (*m*#*ms*)

lemma *omega-sum-0-iff* [*simp*]: *normal ns* $\Longrightarrow$ *omega-sum ns* = *0* $\longleftrightarrow$ *ns* = []
⟨*proof*⟩

lemma *Ord-omega-sum-aux* [*simp*]: *Ord* (*omega-sum-aux k ms*)
⟨*proof*⟩

lemma *Ord-omega-sum*: *Ord* (*omega-sum ms*)
⟨*proof*⟩

lemma *omega-sum-less-ωω* [*intro*]: *omega-sum ms* < *ω*↑*ω*
⟨*proof*⟩

lemma *omega-sum-aux-less*: *omega-sum-aux k ms* < *ω* ↑ *k*
⟨*proof*⟩

lemma *omega-sum-less*: *omega-sum ms* < *ω* ↑ (*length ms*)
⟨*proof*⟩

lemma *omega-sum-ge*: *m* ≠ *0* $\Longrightarrow$ *ω* ↑ (*length ms*) ≤ *omega-sum* (*m*#*ms*)
⟨*proof*⟩

lemma *omega-sum-length-less*:
assumes *normal ns length ms* < *length ns*
shows *omega-sum ms* < *omega-sum ns*
⟨*proof*⟩

lemma *omega-sum-length-leD*:
assumes *omega-sum ms* ≤ *omega-sum ns normal ms*
shows *length ms* ≤ *length ns*
⟨*proof*⟩

lemma *omega-sum-less-eqlen-iff-cases* [*simp*]:
assumes *length ms* = *length ns*
shows *omega-sum* (*m*#*ms*) < *omega-sum* (*n*#*ns*)
$\longleftrightarrow$ *m*<*n* ∨ *m*=*n* ∧ *omega-sum ms* < *omega-sum ns*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *omega-sum-lex-less-iff-cases*:
((*length ms*, *omega-sum* (*m*#*ms*)), (*length ns*, *omega-sum* (*n*#*ns*))) ∈ *less-than* <∗*lex*∗> *VWF*
$\longleftrightarrow$ *length ms* < *length ns*
∨ *length ms* = *length ns* ∧ *m*<*n*
∨ *m*=*n* ∧ ((*length ms*, *omega-sum ms*), (*length ns*, *omega-sum ns*)) ∈ *less-than* <∗*lex*∗> *VWF*
⟨*proof*⟩

lemma *omega-sum-less-iff-cases*:

assumes $m > 0$ $n > 0$
shows *omega-sum* $(m\#ms) <$ *omega-sum* $(n\#ns)$
$\longleftrightarrow$ *length* $ms <$ *length* ns
$\vee$ *length* $ms =$ *length* $ns \wedge m<n$
$\vee$ *length* $ms =$ *length* $ns \wedge m=n \wedge$ *omega-sum* $ms <$ *omega-sum* ns
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *omega-sum-less-iff*:
$((length\ ms,\ omega\text{-}sum\ ms),\ (length\ ns,\ omega\text{-}sum\ ns)) \in less\text{-}than <*lex*> VWF$
$\longleftrightarrow (ms,ns) \in lenlex\ less\text{-}than$
⟨*proof*⟩

lemma *eq-omega-sum-less-iff*:
assumes *length* $ms =$ *length* ns
shows $(omega\text{-}sum\ ms,\ omega\text{-}sum\ ns) \in VWF \longleftrightarrow (ms,ns) \in lenlex\ less\text{-}than$
⟨*proof*⟩

lemma *eq-omega-sum-eq-iff*:
assumes *length* $ms =$ *length* ns
shows *omega-sum* $ms =$ *omega-sum* $ns \longleftrightarrow ms=ns$
⟨*proof*⟩

lemma *inj-omega-sum*: *inj-on omega-sum* $\{l.\ length\ l = n\}$
⟨*proof*⟩

lemma *Ex-omega-sum*: $\gamma \in elts\ (\omega\uparrow n) \Longrightarrow \exists\, ns.\ \gamma = omega\text{-}sum\ ns \wedge length\ ns = n$
⟨*proof*⟩

lemma *omega-sum-drop* [*simp*]: *omega-sum* $(dropWhile\ (\lambda n.\ n=0)\ ns) =$ *omega-sum* ns
⟨*proof*⟩

lemma *normal-drop* [*simp*]: *normal* $(dropWhile\ (\lambda n.\ n=0)\ ns)$
⟨*proof*⟩

lemma *omega-sum-ωω*:
assumes $\gamma \in elts\ (\omega\uparrow\omega)$
obtains *ns* **where** $\gamma =$ *omega-sum ns normal ns*
⟨*proof*⟩

definition *Cantor-ωω* :: $V \Rightarrow$ *nat list*
where $Cantor\text{-}\omega\omega \equiv \lambda x.\ SOME\ ns.\ x = omega\text{-}sum\ ns \wedge normal\ ns$

lemma
assumes $\gamma \in elts\ (\omega\uparrow\omega)$
shows *Cantor-ωω*: *omega-sum* $(Cantor\text{-}\omega\omega\ \gamma) = \gamma$

and *normal-Cantor-ωω*: *normal* (*Cantor-ωω* γ)
⟨*proof*⟩

3.2 Larson's set $W(n)$

definition *WW* :: *nat list set*
where *WW* ≡ {*l*. *strict-sorted l*}

fun *into-WW* :: *nat* ⇒ *nat list* ⇒ *nat list* **where**
into-WW k [] = []
| *into-WW k* (*n*#*ns*) = (*k*+*n*) # *into-WW* (*Suc* (*k*+*n*)) *ns*

fun *from-WW* :: *nat* ⇒ *nat list* ⇒ *nat list* **where**
from-WW k [] = []
| *from-WW k* (*n*#*ns*) = (*n* − *k*) # *from-WW* (*Suc n*) *ns*

lemma *from-into-WW* [*simp*]: *from-WW k* (*into-WW k ns*) = *ns*
⟨*proof*⟩

lemma *inj-into-WW*: *inj* (*into-WW k*)
⟨*proof*⟩

lemma *into-from-WW-aux*:
⟦*strict-sorted ns*; ∀ *n*∈*list.set ns*. *k* ≤ *n*⟧ ⟹ *into-WW k* (*from-WW k ns*) = *ns*
⟨*proof*⟩

lemma *into-from-WW* [*simp*]: *strict-sorted ns* ⟹ *into-WW 0* (*from-WW 0 ns*)
= *ns*
⟨*proof*⟩

lemma *into-WW-imp-ge*: *y* ∈ *List.set* (*into-WW x ns*) ⟹ *x* ≤ *y*
⟨*proof*⟩

lemma *strict-sorted-into-WW*: *strict-sorted* (*into-WW x ns*)
⟨*proof*⟩

lemma *length-into-WW*: *length* (*into-WW x ns*) = *length ns*
⟨*proof*⟩

lemma *WW-eq-range-into*: *WW* = *range* (*into-WW 0*)
⟨*proof*⟩

lemma *into-WW-lenlex-iff*: (*into-WW k ms*, *into-WW k ns*) ∈ *lenlex less-than*
⟷ (*ms*, *ns*) ∈ *lenlex less-than*
⟨*proof*⟩

lemma *wf-llt* [*simp*]: *wf* (*lenlex less-than*) **and** *trans-llt* [*simp*]: *trans* (*lenlex less-than*)
⟨*proof*⟩

lemma *total-llt* [*simp*]: *total-on A* (*lenlex less-than*)
⟨*proof*⟩

lemma *omega-sum-1-less*:
assumes (*ms*,*ns*) ∈ *lenlex less-than* **shows** *omega-sum* (*1*#*ms*) < *omega-sum* (*1*#*ns*)
⟨*proof*⟩

lemma *ordertype-WW-1*: *ordertype WW* (*lenlex less-than*) ≤ *ordertype UNIV* (*lenlex less-than*)
⟨*proof*⟩

lemma *ordertype-WW-2*: *ordertype UNIV* (*lenlex less-than*) ≤ ω↑ω
⟨*proof*⟩

lemma *ordertype-WW-3*: ω↑ω ≤ *ordertype WW* (*lenlex less-than*)
⟨*proof*⟩

lemma *ordertype-WW*: *ordertype WW* (*lenlex less-than*) = ω↑ω
and *ordertype-UNIV-ωω*: *ordertype UNIV* (*lenlex less-than*) = ω↑ω
⟨*proof*⟩

lemma *ordertype-ωω*:
fixes *F* :: *nat* ⇒ *nat list set*
assumes ⋀*j*::*nat*. *ordertype* (*F j*) (*lenlex less-than*) = ω↑*j*
shows *ordertype* (⋃*j*. *F j*) (*lenlex less-than*) = ω↑ω
⟨*proof*⟩

definition *WW-seg* :: *nat* ⇒ *nat list set*
where *WW-seg n* ≡ {*l* ∈ *WW*. *length l* = *n*}

lemma *WW-seg-subset-WW*: *WW-seg n* ⊆ *WW*
⟨*proof*⟩

lemma *WW-eq-UN-WW-seg*: *WW* = (⋃ *n*. *WW-seg n*)
⟨*proof*⟩

lemma *ordertype-list-seg*: *ordertype* {*l*. *length l* = *n*} (*lenlex less-than*) = ω↑*n*
⟨*proof*⟩

lemma *ordertype-WW-seg*: *ordertype* (*WW-seg n*) (*lenlex less-than*) = ω↑*n*
(**is** *ordertype ?W ?R* = ω↑*n*)
⟨*proof*⟩

3.3 Definitions required for the lemmas

3.3.1 Larson's "<"-relation on ordered lists

instantiation *list* :: (*ord*)*ord*
begin

definition $xs < ys \equiv xs \neq [] \wedge ys \neq [] \longrightarrow last\ xs < hd\ ys$ **for** *xs ys* :: *'a list*
definition $xs \leq ys \equiv xs < ys \vee xs = ys$ **for** *xs ys* :: *'a list*

instance
⟨*proof*⟩

end

lemma *less-Nil* [*simp*]: $xs < [] \ [] < xs$
⟨*proof*⟩

lemma *less-sets-imp-list-less*:
assumes *list.set xs* $\ll$ *list.set ys*
shows *xs* < *ys*
⟨*proof*⟩

lemma *less-sets-imp-sorted-list-of-set*:
assumes *A* $\ll$ *B finite A finite B*
shows *list-of A* < *list-of B*
⟨*proof*⟩

lemma *sorted-list-of-set-imp-less-sets*:
assumes *xs* < *ys sorted xs sorted ys*
shows *list.set xs* $\ll$ *list.set ys*
⟨*proof*⟩

lemma *less-list-iff-less-sets*:
assumes *sorted xs sorted ys*
shows *xs* < *ys* $\longleftrightarrow$ *list.set xs* $\ll$ *list.set ys*
⟨*proof*⟩

lemma *strict-sorted-append-iff*:
strict-sorted (*xs* @ *ys*) $\longleftrightarrow$ *xs* < *ys* $\wedge$ *strict-sorted xs* $\wedge$ *strict-sorted ys* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *singleton-less-list-iff*: *sorted xs* $\Longrightarrow$ $[n] < xs \longleftrightarrow \{..n\} \cap$ *list.set xs* $= \{\}$
⟨*proof*⟩

lemma *less-hd-imp-less*: *xs* < [*hd ys*] $\Longrightarrow$ *xs* < *ys*
⟨*proof*⟩

lemma *strict-sorted-concat-I*:

assumes $\bigwedge x.\ x \in \mathit{list.set}\ xs \Longrightarrow \mathit{strict\text{-}sorted}\ x$
$\bigwedge n.\ \mathit{Suc}\ n < \mathit{length}\ xs \Longrightarrow xs!n < xs!\mathit{Suc}\ n$
$xs \in \mathit{lists}\ (-\ \{[]\})$
shows *strict-sorted* (*concat xs*)
⟨*proof*⟩

3.4 Nash Williams for lists

3.4.1 Thin sets of lists

inductive *initial-segment* :: *'a list* ⇒ *'a list* ⇒ *bool*
where *initial-segment xs* (*xs@ys*)

definition *thin* :: *'a list set* ⇒ *bool*
where $\mathit{thin}\ A \equiv \neg\ (\exists x\ y.\ x \in A \wedge y \in A \wedge x \neq y \wedge \mathit{initial\text{-}segment}\ x\ y)$

lemma *initial-segment-ne*:
assumes *initial-segment xs ys xs* ≠ []
shows *ys* ≠ [] ∧ *hd ys* = *hd xs*
⟨*proof*⟩

lemma *take-initial-segment*:
assumes *initial-segment xs ys k* ≤ *length xs*
shows *take k xs* = *take k ys*
⟨*proof*⟩

lemma *initial-segment-length-eq*:
assumes *initial-segment xs ys length xs* = *length ys*
shows *xs* = *ys*
⟨*proof*⟩

lemma *initial-segment-Nil* [*simp*]: *initial-segment* [] *ys*
⟨*proof*⟩

lemma *initial-segment-Cons* [*simp*]: *initial-segment* (*x#xs*) (*y#ys*) ⟷ *x=y* ∧ *initial-segment xs ys*
⟨*proof*⟩

lemma *init-segment-iff-initial-segment*:
assumes *strict-sorted xs strict-sorted ys*
shows *init-segment* (*list.set xs*) (*list.set ys*) ⟷ *initial-segment xs ys* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

theorem *Nash-Williams-WW*:
fixes *h* :: *nat list* ⇒ *nat*
assumes *infinite M* **and** *h*: *h* ‘ {*l* ∈ *A*. *List.set l* ⊆ *M*} ⊆ {..<*2*} **and** *thin A A* ⊆ *WW*
obtains *i N* **where** *i* < *2 infinite N N* ⊆ *M h* ‘ {*l* ∈ *A*. *List.set l* ⊆ *N*} ⊆ {*i*}
⟨*proof*⟩

3.5 Specialised functions on lists

lemma *mem-lists-non-Nil*: $xss \in lists\ (-\ \{[]\}) \longleftrightarrow (\forall x \in list.set\ xss.\ x \neq [])$
⟨*proof*⟩

fun *acc-lengths* :: $nat \Rightarrow 'a\ list\ list \Rightarrow nat\ list$
where $acc\text{-}lengths\ acc\ [] = []$
| $acc\text{-}lengths\ acc\ (l\#ls) = (acc + length\ l)\ \#\ acc\text{-}lengths\ (acc + length\ l)\ ls$

lemma *length-acc-lengths* [*simp*]: $length\ (acc\text{-}lengths\ acc\ ls) = length\ ls$
⟨*proof*⟩

lemma *acc-lengths-eq-Nil-iff* [*simp*]: $acc\text{-}lengths\ acc\ ls = [] \longleftrightarrow ls = []$
⟨*proof*⟩

lemma *set-acc-lengths*:
assumes $ls \in lists\ (-\ \{[]\})$ **shows** $list.set\ (acc\text{-}lengths\ acc\ ls) \subseteq \{acc<..\}$
⟨*proof*⟩

Useful because *acc-lengths.simps* will sometimes be deleted from the simpset.

lemma *hd-acc-lengths* [*simp*]: $hd\ (acc\text{-}lengths\ acc\ (l\#ls)) = acc + length\ l$
⟨*proof*⟩

lemma *last-acc-lengths* [*simp*]:
$ls \neq [] \Longrightarrow last\ (acc\text{-}lengths\ acc\ ls) = acc + sum\text{-}list\ (map\ length\ ls)$
⟨*proof*⟩

lemma *nth-acc-lengths* [*simp*]:
$[\![ls \neq []; k < length\ ls]\!] \Longrightarrow acc\text{-}lengths\ acc\ ls\ !\ k = acc + sum\text{-}list\ (map\ length\ (take\ (Suc\ k)\ ls))$
⟨*proof*⟩

lemma *acc-lengths-plus*: $acc\text{-}lengths\ (m+n)\ as = map\ ((+)m)\ (acc\text{-}lengths\ n\ as)$
⟨*proof*⟩

lemma *acc-lengths-shift*: $NO\text{-}MATCH\ 0\ acc \Longrightarrow acc\text{-}lengths\ acc\ as = map\ ((+)acc)\ (acc\text{-}lengths\ 0\ as)$
⟨*proof*⟩

lemma *length-concat-acc-lengths*:
$ls \neq [] \Longrightarrow k + length\ (concat\ ls) \in list.set\ (acc\text{-}lengths\ k\ ls)$
⟨*proof*⟩

lemma *strict-sorted-acc-lengths*:
assumes $ls \in lists\ (-\ \{[]\})$ **shows** $strict\text{-}sorted\ (acc\text{-}lengths\ acc\ ls)$
⟨*proof*⟩

lemma *acc-lengths-append*:
$acc\text{-}lengths\ acc\ (xs\ @\ ys)$

$= acc\text{-}lengths\ acc\ xs\ @\ acc\text{-}lengths\ (acc + sum\text{-}list\ (map\ length\ xs))\ ys$
⟨*proof*⟩

lemma *length-concat-ge*:
assumes $as \in lists\ (- \{[]\})$
shows $length\ (concat\ as) \geq length\ as$
⟨*proof*⟩

fun $interact :: 'a\ list\ list \Rightarrow 'a\ list\ list \Rightarrow 'a\ list$
where
$interact\ []\ ys = concat\ ys$
$| interact\ xs\ [] = concat\ xs$
$| interact\ (x\#xs)\ (y\#ys) = x\ @\ y\ @\ interact\ xs\ ys$

lemma (**in** *monoid-add*) *length-interact*:
$length\ (interact\ xs\ ys) = sum\text{-}list\ (map\ length\ xs) + sum\text{-}list\ (map\ length\ ys)$
⟨*proof*⟩

lemma *length-interact-ge*:
assumes $xs \in lists\ (- \{[]\})\ ys \in lists\ (- \{[]\})$
shows $length\ (interact\ xs\ ys) \geq length\ xs + length\ ys$
⟨*proof*⟩

lemma *set-interact* [*simp*]:
shows $list.set\ (interact\ xs\ ys) = list.set\ (concat\ xs) \cup list.set\ (concat\ ys)$
⟨*proof*⟩

lemma *interact-eq-Nil-iff* [*simp*]:
assumes $xs \in lists\ (- \{[]\})\ ys \in lists\ (- \{[]\})$
shows $interact\ xs\ ys = [] \longleftrightarrow xs{=}[] \wedge ys{=}[]$
⟨*proof*⟩

lemma *interact-sing* [*simp*]: $interact\ [x]\ ys = x\ @\ concat\ ys$
⟨*proof*⟩

lemma *hd-interact*: $[\![xs \neq []; hd\ xs \neq []]\!] \Longrightarrow hd\ (interact\ xs\ ys) = hd\ (hd\ xs)$
⟨*proof*⟩

lemma *acc-lengths-concat-injective*:
assumes $concat\ as' = concat\ as\ acc\text{-}lengths\ n\ as' = acc\text{-}lengths\ n\ as$
shows $as' = as$
⟨*proof*⟩

lemma *acc-lengths-interact-injective*:
assumes $interact\ as'\ bs' = interact\ as\ bs\ acc\text{-}lengths\ a\ as' = acc\text{-}lengths\ a\ as$
$acc\text{-}lengths\ b\ bs' = acc\text{-}lengths\ b\ bs$

shows $as' = as \wedge bs' = bs$
⟨*proof*⟩

lemma *strict-sorted-interact-I*:
assumes *length ys* ≤ *length xs length xs* ≤ *Suc* (*length ys*)
⋀*x*. *x* ∈ *list.set xs* ⟹ *strict-sorted x*
⋀*y*. *y* ∈ *list.set ys* ⟹ *strict-sorted y*
⋀*n*. *n* < *length ys* ⟹ *xs*!*n* < *ys*!*n*
⋀*n*. *Suc n* < *length xs* ⟹ *ys*!*n* < *xs*!*Suc n*
assumes *xs* ∈ *lists* (− {[]}) *ys* ∈ *lists* (− {[]})
shows *strict-sorted* (*interact xs ys*)
⟨*proof*⟩

3.6 Forms and interactions

3.6.1 Forms

inductive *Form-Body* :: [*nat*, *nat*, *nat list*, *nat list*, *nat list*] ⇒ *bool*
where *Form-Body ka kb xs ys zs*
if *length xs* < *length ys xs* = *concat* (*a*#*as*) *ys* = *concat* (*b*#*bs*)
a#*as* ∈ *lists* (− {[]}) *b*#*bs* ∈ *lists* (− {[]})
length (*a*#*as*) = *ka length* (*b*#*bs*) = *kb*
c = *acc-lengths 0* (*a*#*as*)
d = *acc-lengths 0* (*b*#*bs*)
zs = *concat* [*c*, *a*, *d*, *b*] @ *interact as bs*
strict-sorted zs

inductive *Form* :: [*nat*, *nat list set*] ⇒ *bool*
where *Form 0* {*xs*,*ys*} **if** *length xs* = *length ys xs* ≠ *ys*
| *Form* (*2*∗*k*−*1*) {*xs*,*ys*} **if** *Form-Body k k xs ys zs k* > *0*
| *Form* (*2*∗*k*) {*xs*,*ys*} **if** *Form-Body* (*Suc k*) *k xs ys zs k* > *0*

inductive-cases *Form-0-cases-raw*: *Form 0 u*

lemma *Form-elim-upair*:
assumes *Form l U*
obtains *xs ys* **where** *xs* ≠ *ys U* = {*xs*,*ys*} *length xs* ≤ *length ys*
⟨*proof*⟩

lemma assumes *Form-Body ka kb xs ys zs*
shows *Form-Body-WW*: *zs* ∈ *WW*
and *Form-Body-nonempty*: *length zs* > *0*
and *Form-Body-length*: *length xs* < *length ys*
⟨*proof*⟩

lemma *form-cases*:
fixes *l*::*nat*

obtains (*zero*) $l = 0$ | (*nz*) *ka kb* **where** $l = ka+kb - 1$ $0 < kb$ $kb \leq ka$ $ka \leq Suc\ kb$
⟨*proof*⟩

3.6.2 Interactions

lemma *interact*:
assumes *Form l U l>0*
obtains *ka kb xs ys zs* **where** $l = ka+kb - 1$ $U = \{xs,ys\}$ *Form-Body ka kb xs ys zs* $0 < kb$ $kb \leq ka$ $ka \leq Suc\ kb$
⟨*proof*⟩

definition *inter-scheme* :: *nat* ⇒ *nat list set* ⇒ *nat list*
where *inter-scheme l U* ≡
SOME zs. ∃ *k xs ys. U* = {*xs*,*ys*} ∧
(*l* = *2*k*−*1* ∧ *Form-Body k k xs ys zs* ∨ *l* = *2*k* ∧ *Form-Body* (*Suc k*) *k xs ys zs*)

lemma *inter-scheme*:
assumes *Form l U l>0*
obtains *ka kb xs ys* **where** $l = ka+kb - 1$ $U = \{xs,ys\}$ *Form-Body ka kb xs ys* (*inter-scheme l U*) $0 < kb$ $kb \leq ka$ $ka \leq Suc\ kb$
⟨*proof*⟩

lemma *inter-scheme-strict-sorted*:
assumes *Form l U l>0*
shows *strict-sorted* (*inter-scheme l U*)
⟨*proof*⟩

lemma *inter-scheme-simple*:
assumes *Form l U l>0*
shows *inter-scheme l U* ∈ *WW* ∧ *length* (*inter-scheme l U*) > *0*
⟨*proof*⟩

3.6.3 Injectivity of interactions

proposition *inter-scheme-injective*:
assumes *Form l U Form l U′ l* > *0* **and** *eq*: *inter-scheme l U′* = *inter-scheme l U*
shows $U' = U$
⟨*proof*⟩

lemma *strict-sorted-interact-imp-concat*:
strict-sorted (*interact as bs*) ⟹ *strict-sorted* (*concat as*) ∧ *strict-sorted* (*concat bs*)
⟨*proof*⟩

lemma *strict-sorted-interact-hd*:
⟦*strict-sorted* (*interact cs ds*); *cs* ≠ []; *ds* ≠ []; *hd cs* ≠ []; *hd ds* ≠ []⟧
⟹ *hd* (*hd cs*) < *hd* (*hd ds*)
⟨*proof*⟩

the lengths of the two lists can differ by one

proposition *interaction-scheme-unique-aux*:
assumes *concat as* = *concat as′* **and** *ys′*: *concat bs* = *concat bs′*
and *as* ∈ *lists* (− {[]}) *bs* ∈ *lists* (− {[]})
and *strict-sorted* (*interact as bs*)
and *length bs* ≤ *length as length as* ≤ *Suc* (*length bs*)
and *as′* ∈ *lists* (− {[]}) *bs′* ∈ *lists* (− {[]})
and *strict-sorted* (*interact as′ bs′*)
and *length bs′* ≤ *length as′ length as′* ≤ *Suc* (*length bs′*)
and *length as* = *length as′ length bs* = *length bs′*
shows *as* = *as′* ∧ *bs* = *bs′*
⟨*proof*⟩

proposition *Form-Body-unique*:
assumes *Form-Body ka kb xs ys zs Form-Body ka kb xs ys zs′* **and** *kb* ≤ *ka ka* ≤ *Suc kb*
shows *zs′* = *zs*
⟨*proof*⟩

lemma *Form-Body-imp-inter-scheme*:
assumes *FB*: *Form-Body ka kb xs ys zs* **and** *0* < *kb kb* ≤ *ka ka* ≤ *Suc kb*
shows *zs* = *inter-scheme* ((*ka*+*kb*) − *Suc 0*) {*xs*,*ys*}
⟨*proof*⟩

3.7 For Lemma 3.8 AND PROBABLY 3.7

definition *grab* :: *nat set* ⇒ *nat* ⇒ *nat set* × *nat set*
where *grab N n* ≡ (*N* ∩ *enumerate N* ‘ {..<*n*}, *N* ∩ {*enumerate N n*..})

lemma *grab-0* [*simp*]: *grab N 0* = ({}, *N*)
⟨*proof*⟩

lemma *less-sets-grab*:
infinite N ⟹ *fst* (*grab N n*) ≪ *snd* (*grab N n*)
⟨*proof*⟩

lemma *finite-grab* [*iff*]: *finite* (*fst* (*grab N n*))
⟨*proof*⟩

lemma *card-grab* [*simp*]:
assumes *infinite N* **shows** *card* (*fst* (*grab N n*)) = *n*
⟨*proof*⟩

lemma *fst-grab-subset*: *fst* (*grab N n*) ⊆ *N*
⟨*proof*⟩

lemma *snd-grab-subset*: *snd* (*grab N n*) ⊆ *N*
⟨*proof*⟩

lemma *grab-Un-eq*:
assumes *infinite N* **shows** *fst* (*grab N n*) ∪ *snd* (*grab N n*) = *N*
⟨*proof*⟩

lemma *finite-grab-iff* [*simp*]: *finite* (*snd* (*grab N n*)) ⟷ *finite N*
⟨*proof*⟩

lemma *grab-eqD*:
⟦*grab N n* = (*A*,*M*); *infinite N*⟧
⟹ *A* ≪ *M* ∧ *finite A* ∧ *card A* = *n* ∧ *infinite M* ∧ *A* ⊆ *N* ∧ *M* ⊆ *N*
⟨*proof*⟩

lemma *less-sets-fst-grab*: *A* ≪ *N* ⟹ *A* ≪ *fst* (*grab N n*)
⟨*proof*⟩

Possibly redundant, given *grab*

definition *nxt* **where** *nxt* ≡ λ*N*. λ*n*::*nat*. *N* ∩ {*n*<..}

lemma *infinite-nxtN*: *infinite N* ⟹ *infinite* (*nxt N n*)
⟨*proof*⟩

lemma *nxt-subset*: *nxt N n* ⊆ *N*
⟨*proof*⟩

lemma *nxt-subset-greaterThan*: *m* ≤ *n* ⟹ *nxt N n* ⊆ {*m*<..}
⟨*proof*⟩

lemma *nxt-subset-atLeast*: *m* ≤ *n* ⟹ *nxt N n* ⊆ {*m*..}
⟨*proof*⟩

lemma *enum-nxt-ge*: *infinite N* ⟹ *a* ≤ *enum* (*nxt N a*) *n*
⟨*proof*⟩

lemma *inj-enum-nxt*: *infinite N* ⟹ *inj-on* (*enum* (*nxt N a*)) *A*
⟨*proof*⟩

3.8 Larson's Lemma 3.11

Again from Jean A. Larson, A short proof of a partition theorem for the ordinal ω^ω. *Annals of Mathematical Logic*, 6:129–145, 1973.

lemma *lemma-3-11*:
assumes *l* > *0*

shows *thin* (*inter-scheme l* ‘ {*U*. *Form l U*})
⟨*proof*⟩

3.9 Larson's Lemma 3.6

proposition *lemma-3-6*:
fixes *g* :: *nat list set* ⇒ *nat*
assumes *g*: *g* ∈ [*WW*]^2 → {*0*,*1*}
obtains *N j* **where** *infinite N*
and ⋀*k u*. ⟦*k* > *0*; *u* ∈ [*WW*]^2; *Form k u*; [*enum N k*] < *inter-scheme k u*; *List.set* (*inter-scheme k u*) ⊆ *N*⟧ ⟹ *g u* = *j k*
⟨*proof*⟩

3.10 Larson's Lemma 3.7

3.10.1 Preliminaries

Analogous to *ordered-nsets-2-eq*, but without type classes

lemma *total-order-nsets-2-eq*:
assumes *tot*: *total-on A r* **and** *irr*: *irrefl r*
shows *nsets A 2* = {{*x*,*y*} | *x y*. *x* ∈ *A* ∧ *y* ∈ *A* ∧ (*x*,*y*) ∈ *r*}
(**is** *- = ?rhs*)
⟨*proof*⟩

lemma *lenlex-nsets-2-eq*: *nsets A 2* = {{*x*,*y*} | *x y*. *x* ∈ *A* ∧ *y* ∈ *A* ∧ (*x*,*y*) ∈ *lenlex less-than*}
⟨*proof*⟩

lemma *sum-sorted-list-of-set-map*: *finite I* ⟹ *sum-list* (*map f* (*list-of I*)) = *sum f I*
⟨*proof*⟩

lemma *sorted-list-of-set-UN-eq-concat*:
assumes *I*: *strict-mono-sets I f finite I* **and** *fin*: ⋀*i*. *finite* (*f i*)
shows *list-of* (⋃ *i* ∈ *I*. *f i*) = *concat* (*map* (*list-of* ∘ *f*) (*list-of I*))
⟨*proof*⟩

3.10.2 Lemma 3.7 of Jean A. Larson, ibid.

proposition *lemma-3-7*:
assumes *infinite N l* > *0*
obtains *M* **where** *M* ∈ [*WW*]^m
⋀*U*. *U* ∈ [*M*]^2 ⟹ *Form l U* ∧ *List.set* (*inter-scheme l U*) ⊆ *N*
⟨*proof*⟩

3.11 Larson's Lemma 3.8

3.11.1 Primitives needed for the inductive construction of b

definition *IJ* **where** $IJ \equiv \lambda k.\ Sigma\ \{..k\}\ (\lambda j{::}nat.\ \{..<j\})$

lemma *IJ-iff*: $u \in IJ\ k \longleftrightarrow (\exists j\ i.\ u = (j,i) \wedge i<j \wedge j \leq k)$
⟨*proof*⟩

lemma *finite-IJ*: *finite* $(IJ\ k)$
⟨*proof*⟩

fun *prev* **where**
$prev\ 0\ 0 = None$
| $prev\ (Suc\ 0)\ 0 = None$
| $prev\ (Suc\ j)\ 0 = Some\ (j,\ j - Suc\ 0)$
| $prev\ j\ (Suc\ i) = Some\ (j,i)$

lemma *prev-eq-None-iff*: $prev\ j\ i = None \longleftrightarrow j \leq Suc\ 0 \wedge i = 0$
⟨*proof*⟩

lemma *prev-pair-less*:
$prev\ j\ i = Some\ ji' \Longrightarrow (ji',\ (j,i)) \in pair\text{-}less$
⟨*proof*⟩

lemma *prev-Some-less*: $[\![prev\ j\ i = Some\ (j',i');\ i \leq j]\!] \Longrightarrow i' < j'$
⟨*proof*⟩

lemma *prev-maximal*:
$[\![prev\ j\ i = Some\ (j',i');\ (ji'',\ (j,i)) \in pair\text{-}less;\ ji'' \in IJ\ k]\!]$
$\Longrightarrow (ji'',\ (j',i')) \in pair\text{-}less \vee ji'' = (j',i')$
⟨*proof*⟩

lemma *pair-less-prev*:
assumes $(u,\ (j,i)) \in pair\text{-}less\ u \in IJ\ k$
shows $prev\ j\ i = Some\ u \vee (\exists x.\ (u,\ x) \in pair\text{-}less \wedge prev\ j\ i = Some\ x)$
⟨*proof*⟩

3.11.2 Special primitives for the ordertype proof

definition $USigma :: 'a\ set\ set \Rightarrow ('a\ set \Rightarrow 'a\ set) \Rightarrow 'a\ set\ set$
where $USigma\ \mathcal{A}\ B \equiv \bigcup X{\in}\mathcal{A}.\ \bigcup y \in B\ X.\ \{insert\ y\ X\}$

definition *usplit*
where $usplit\ f\ A \equiv f\ (A - \{Max\ A\})\ (Max\ A)$

lemma *USigma-empty* [*simp*]: $USigma\ \{\}\ B = \{\}$
⟨*proof*⟩

lemma *USigma-iff*:

assumes $\bigwedge I\ j.\ I \in \mathcal{I} \Longrightarrow I \ll J\ I \wedge \mathit{finite}\ I$
shows $x \in \mathit{USigma}\ \mathcal{I}\ J \longleftrightarrow \mathit{usplit}\ (\lambda I\ j.\ I \in \mathcal{I} \wedge j \in J\ I \wedge x = \mathit{insert}\ j\ I)\ x$
⟨*proof*⟩

proposition *ordertype-append-image-IJ*:
assumes *lenB* [*simp*]: $\bigwedge i\ j.\ i \in \mathcal{I} \Longrightarrow j \in J\ i \Longrightarrow \mathit{length}\ (B\ j) = c$
and *AB*: $\bigwedge i\ j.\ i \in \mathcal{I} \Longrightarrow j \in J\ i \Longrightarrow A\ i < B\ j$
and *IJ*: $\bigwedge i.\ i \in \mathcal{I} \Longrightarrow i \ll J\ i \wedge \mathit{finite}\ i$
and β: $\bigwedge i.\ i \in \mathcal{I} \Longrightarrow \mathit{ordertype}\ (B\ `\ J\ i)\ (\mathit{lenlex}\ \mathit{less\text{-}than}) = \beta$
and *A*: *inj-on A* $\mathcal{I}$
shows $\mathit{ordertype}\ (\mathit{usplit}\ (\lambda i\ j.\ A\ i\ @\ B\ j)\ `\ \mathit{USigma}\ \mathcal{I}\ J)\ (\mathit{lenlex}\ \mathit{less\text{-}than})$
$= \beta * \mathit{ordertype}\ (A\ `\ \mathcal{I})\ (\mathit{lenlex}\ \mathit{less\text{-}than})$
(**is** $\mathit{ordertype}\ ?AB\ ?R = \text{-} * ?\alpha$)
⟨*proof*⟩

3.11.3 The final part of 3.8, where two sequences are merged

inductive *merge* :: [*nat list list*,*nat list list*,*nat list list*,*nat list list*] ⇒ *bool*
where *NullNull*: *merge* [] [] [] []
| *Null*: $as \neq [] \Longrightarrow \mathit{merge}\ as\ []\ [\mathit{concat}\ as]\ []$
| *App*: ⟦*as1* ≠ []; *bs1* ≠ []; *concat as1* < *concat bs1*; *concat bs1* < *concat as2*; *merge as2 bs2 as bs*⟧
$\Longrightarrow \mathit{merge}\ (as1@as2)\ (bs1@bs2)\ (\mathit{concat}\ as1\ \#\ as)\ (\mathit{concat}\ bs1\ \#\ bs)$

inductive-simps *Null1* [*simp*]: *merge* [] *bs us vs*
inductive-simps *Null2* [*simp*]: *merge as* [] *us vs*

lemma *merge-single*:
⟦*concat as* < *concat bs*; *concat as* ≠ []; *concat bs* ≠ []⟧ ⟹ *merge as bs* [*concat as*] [*concat bs*]
⟨*proof*⟩

lemma *merge-length1-nonempty*:
assumes *merge as bs us vs as* ∈ *lists* (− {[]})
shows *us* ∈ *lists* (− {[]})
⟨*proof*⟩

lemma *merge-length2-nonempty*:
assumes *merge as bs us vs bs* ∈ *lists* (− {[]})
shows *vs* ∈ *lists* (− {[]})
⟨*proof*⟩

lemma *merge-length1-gt-0*:
assumes *merge as bs us vs as* ≠ []
shows *length us* > *0*
⟨*proof*⟩

lemma *merge-length-le*:
assumes *merge as bs us vs*
shows *length vs* ≤ *length us*
⟨*proof*⟩

lemma *merge-length-le-Suc*:
assumes *merge as bs us vs*
shows *length us* ≤ *Suc* (*length vs*)
⟨*proof*⟩

lemma *merge-length-less2*:
assumes *merge as bs us vs*
shows *length vs* ≤ *length as*
⟨*proof*⟩

lemma *merge-preserves*:
assumes *merge as bs us vs*
shows *concat as* = *concat us* ∧ *concat bs* = *concat vs*
⟨*proof*⟩

lemma *merge-interact*:
assumes *merge as bs us vs strict-sorted* (*concat as*) *strict-sorted* (*concat bs*)
bs ∈ *lists* (− {[]})
shows *strict-sorted* (*interact us vs*)
⟨*proof*⟩

lemma *acc-lengths-merge1*:
assumes *merge as bs us vs*
shows *list.set* (*acc-lengths k us*) ⊆ *list.set* (*acc-lengths k as*)
⟨*proof*⟩

lemma *acc-lengths-merge2*:
assumes *merge as bs us vs*
shows *list.set* (*acc-lengths k vs*) ⊆ *list.set* (*acc-lengths k bs*)
⟨*proof*⟩

lemma *length-hd-le-concat*:
assumes *as* ≠ [] **shows** *length* (*hd as*) ≤ *length* (*concat as*)
⟨*proof*⟩

lemma *length-hd-merge2*:
assumes *merge as bs us vs*
shows *length* (*hd bs*) ≤ *length* (*hd vs*)
⟨*proof*⟩

lemma *merge-less-sets-hd*:
assumes *merge as bs us vs strict-sorted* (*concat as*) *strict-sorted* (*concat bs*) *bs* ∈ *lists* (− {[]})

shows *list.set* (*hd us*) $\ll$ *list.set* (*concat vs*)
⟨*proof*⟩

lemma *set-takeWhile*:
assumes *strict-sorted* (*concat as*) *as* ∈ *lists* (− {[]})
shows *list.set* (*takeWhile* (λ*x*. *x* < *y*) *as*) = {*x* ∈ *list.set as*. *x* < *y*}
⟨*proof*⟩

proposition *merge-exists*:
assumes *strict-sorted* (*concat as*) *strict-sorted* (*concat bs*)
as ∈ *lists* (− {[]}) *bs* ∈ *lists* (− {[]})
hd as < *hd bs as* ≠ [] *bs* ≠ []
and *disj*: ⋀*a b*. ⟦*a* ∈ *list.set as*; *b* ∈ *list.set bs*⟧ ⟹ *a*<*b* ∨ *b*<*a*
shows ∃ *us vs*. *merge as bs us vs*
⟨*proof*⟩

3.11.4 Actual proof of Larson's Lemma 3.8

proposition *lemma-3-8*:
assumes *infinite N*
obtains *X* **where** *X* ⊆ *WW ordertype X* (*lenlex less-than*) = ω↑ω
⋀*u*. *u* ∈ [*X*]^2 ⟹
∃ *l*. *Form l u* ∧ (*l* > *0* ⟶ [*enum N l*] < *inter-scheme l u* ∧ *List.set* (*inter-scheme l u*) ⊆ *N*)
⟨*proof*⟩

3.12 The main partition theorem for ω ↑ ω

definition *iso-ll* **where** *iso-ll A B* ≡ *iso* (*lenlex less-than* ∩ (*A*×*A*)) (*lenlex less-than* ∩ (*B*×*B*))

corollary *ordertype-eq-ordertype-iso-ll*:
assumes *Field* (*Restr* (*lenlex less-than*) *A*) = *A Field* (*Restr* (*lenlex less-than*) *B*) = *B*
shows (*ordertype A* (*lenlex less-than*) = *ordertype B* (*lenlex less-than*))
⟷ (∃ *f*. *iso-ll A B f*)
⟨*proof*⟩

theorem *partition-ωω-aux*:
assumes α ∈ *elts* ω
shows *partn-lst* (*lenlex less-than*) *WW* [ω↑ω,α] *2* (**is** *partn-lst ?R WW* [ω↑ω,α] *2*)
⟨*proof*⟩

Theorem 3.1 of Jean A. Larson, ibid.

theorem *partition-ωω*: α ∈ *elts* ω ⟹ *partn-lst-VWF* (ω↑ω) [ω↑ω,α] *2*
⟨*proof*⟩

end

4 Acknowledgements

The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council. Many thanks to Mirna Džamonja (who suggested the project) and Angeliki Koutsoukou-Argyraki for assistance at tricky moments.

References

[1] P. Erdös and E. C. Milner. A theorem in the partition calculus. *Canadian Mathematical Bulletin*, 15(4):501–505, Dec. 1972.

[2] P. Erdös and E. C. Milner. A theorem in the partition calculus corrigendum. *Canadian Mathematical Bulletin*, 17(2):305, June 1974.

[3] J. A. Larson. A short proof of a partition theorem for the ordinal ω^ω. *Annals of Mathematical Logic*, 6(2):129–145, Dec. 1973.