

# A Partition Theorem for the Ordinal $\omega^\omega$

Lawrence C. Paulson

March 17, 2025

## Abstract

The theory of partition relations concerns generalisations of Ramsey's theorem. For any ordinal  $\alpha$ , write  $\alpha \rightarrow (\alpha, m)^2$  if for each function  $f$  from unordered pairs of elements of  $\alpha$  into  $\{0, 1\}$ , either there is a subset  $X \subseteq \alpha$  order-isomorphic to  $\alpha$  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  $\omega \rightarrow (\omega, \omega)^2$ , or if we restrict  $m$  to the positive integers as above, then  $\omega \rightarrow (\omega, m)^2$  for all  $m$  [3].

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

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

<b>1</b>	<b>Library additions</b>	<b>2</b>
1.1	Other material . . . . .	4
1.2	The list-of function . . . . .	5
1.3	Monotonic enumeration of a countably infinite set . . . . .	6
<b>2</b>	<b>Ordinal Partitions</b>	<b>9</b>
2.1	Ordinal Partitions: Definitions . . . . .	9
2.2	Relating partition properties on $VWF$ to the general case . .	13
2.3	Simple consequences of the definitions . . . . .	16
2.4	Specker's theorem . . . . .	18
2.5	Erds-Milner theorem . . . . .	28
<b>3</b>	<b>An ordinal partition theorem by Jean A. Larson</b>	<b>56</b>
3.1	Cantor normal form for ordinals below $\omega \uparrow \omega$ . . . . .	56
3.2	Larson's set $W(n)$ . . . . .	60
3.3	Definitions required for the lemmas . . . . .	63

3.3.1	Larson's " $<$ "-relation on ordered lists . . . . .	63
3.4	Nash Williams for lists . . . . .	64
3.4.1	Thin sets of lists . . . . .	64
3.5	Specialised functions on lists . . . . .	67
3.6	Forms and interactions . . . . .	70
3.6.1	Forms . . . . .	70
3.6.2	Interactions . . . . .	71
3.6.3	Injectivity of interactions . . . . .	72
3.7	For Lemma 3.8 AND PROBABLY 3.7 . . . . .	79
3.8	Larson's Lemma 3.11 . . . . .	81
3.9	Larson's Lemma 3.6 . . . . .	82
3.10	Larson's Lemma 3.7 . . . . .	85
3.10.1	Preliminaries . . . . .	85
3.10.2	Lemma 3.7 of Jean A. Larson, <i>ibid.</i> . . . . .	87
3.11	Larson's Lemma 3.8 . . . . .	105
3.11.1	Primitives needed for the inductive construction of $b$ . . . . .	105
3.11.2	Special primitives for the ordertype proof . . . . .	106
3.11.3	The final part of 3.8, where two sequences are merged . . . . .	111
3.11.4	Actual proof of Larson's Lemma 3.8 . . . . .	116
3.12	The main partition theorem for $\omega \uparrow \omega$ . . . . .	140

**4 Acknowledgements** **148**

**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
  using  $i$ 
proof (induction i)
  case 0
  have  $(LEAST\ i.\ i \in S \wedge i \neq x) = (LEAST\ i.\ i \in S)$ 
  proof (rule Least-equality)
    have  $\exists t.\ t \in S \wedge t \neq x$ 
      using 0  $\langle finite\ S \rangle$  finite-enumerate-in-set by blast
    then show  $(LEAST\ i.\ i \in S) \in S \wedge (LEAST\ i.\ i \in S) \neq x$ 
      by (metis 0.prem(2) LeastI enumerate-0 not-less-Least)
  qed (simp add: Least-le)
then show ?case
  by (auto simp: enumerate-0)

```

**next**  
**case** (*Suc i*)  
**then have** *x*: *enumerate S i < x*  
**by** (*meson enumerate-step finite-enumerate-step less-trans*)  
**have** *cardSx*: *Suc i < card (S - {x})* **and** *i < card S*  
**using** *Suc*  $\langle$ *finite S* $\rangle$  *card-Diff-singleton-if*[*of S*] *finite-enumerate-Ex* **by** *fast-force+*  
**have** (*LEAST s. s ∈ S ∧ s ≠ x ∧ enumerate (S - {x}) i < s*) = (*LEAST s. s ∈ S ∧ enumerate S i < s*)  
**(is - = ?r)**  
**proof** (*intro Least-equality conjI*)  
**show** *?r ∈ S*  
**by** (*metis (lifting) LeastI Suc.prem(1) assms(1) finite-enumerate-in-set finite-enumerate-step*)  
**show** *?r ≠ x*  
**using** *Suc.prem not-less-Least* [*of - λt. t ∈ S ∧ enumerate S i < t*]  
 $\langle$ *finite S* $\rangle$  *finite-enumerate-in-set finite-enumerate-step* **by** *blast*  
**show** *enumerate (S - {x}) i < ?r*  
**by** (*metis (full-types) Suc.IH Suc.prem(1) <i < card S> enumerate-Suc''*  
*enumerate-step finite-enumerate-Suc'' finite-enumerate-step x*)  
**show**  $\bigwedge y. y ∈ S ∧ y ≠ x ∧ enumerate (S - {x}) i < y \implies ?r \leq y$   
**by** (*simp add: Least-le Suc.IH <i < card S> x*)  
**qed**  
**then show** *?case*  
**using** *Suc assms* **by** (*simp add: finite-enumerate-Suc'' cardSx*)  
**qed**

**lemma** *hd-lex*:  $\llbracket hd\ ms < hd\ ns; length\ ms = length\ ns; ns \neq [] \rrbracket \implies (ms, ns) \in lex\ less-than$   
**by** (*metis hd-Cons-tl length-0-conv less-than-iff lexord-cons-cons lexord-lex*)

**lemma** *sorted-hd-le*:  
**assumes** *sorted xs x ∈ list.set xs*  
**shows** *hd xs ≤ x*  
**using** *assms* **by** (*induction xs*) (*auto simp: less-imp-le*)

**lemma** *sorted-le-last*:  
**assumes** *sorted xs x ∈ list.set xs*  
**shows** *x ≤ last xs*  
**using** *assms* **by** (*induction xs*) (*auto simp: less-imp-le*)

**lemma** *hd-list-of*:  
**assumes** *finite A A ≠ {}*  
**shows** *hd (sorted-list-of-set A) = Min A*  
**proof** (*rule antisym*)  
**have** *Min A ∈ A*  
**by** (*simp add: assms*)  
**then show** *hd (sorted-list-of-set A) ≤ Min A*  
**by** (*simp add: sorted-hd-le <finite A>*)

```

next
  show  $Min\ A \leq hd\ (sorted\ list\ of\ set\ A)$ 
  by (metis Min-le assms hd-in-set set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff)
qed

```

```

lemma sorted-hd-le-last:
  assumes sorted xs xs  $\neq []$ 
  shows  $hd\ xs \leq last\ xs$ 
  using assms by (simp add: sorted-hd-le)

```

```

lemma sorted-list-of-set-set-of [simp]:  $strict\ sorted\ l \implies sorted\ list\ of\ set\ (list.set\ l) = l$ 
  by (simp add: strict-sorted-equal)

```

```

lemma range-strict-mono-ext:
  fixes  $f :: nat \Rightarrow 'a :: linorder$ 
  assumes  $eq: range\ f = range\ g$ 
  and  $sm: strict\ mono\ f\ strict\ mono\ g$ 
  shows  $f = g$ 

```

```

proof
  fix  $n$ 
  show  $f\ n = g\ n$ 
  proof (induction  $n$  rule: less-induct)
    case (less  $n$ )
    obtain  $x\ y$  where  $xy: f\ n = g\ y\ f\ x = g\ n$ 
    by (metis eq imageE rangeI)
    then have  $n = y$ 
    by (metis (no-types) less.IH neq-iff sm strict-mono-less  $xy$ )
    then show ?case using  $xy$  by auto
  qed
qed

```

## 1.1 Other material

```

definition strict-mono-sets :: [ $'a :: order\ set, 'a :: order \Rightarrow 'b :: order\ set$ ]  $\Rightarrow bool$  where
  strict-mono-sets  $A\ f \equiv \forall x \in A. \forall y \in A. x < y \longrightarrow less\ sets\ (f\ x)\ (f\ y)$ 

```

```

lemma strict-mono-setsD:
  assumes strict-mono-sets  $A\ f\ x < y\ x \in A\ y \in A$ 
  shows less-sets  $(f\ x)\ (f\ y)$ 
  using assms by (auto simp: strict-mono-sets-def)

```

```

lemma strict-mono-sets-imp-disjoint:
  fixes  $A :: 'a :: linorder\ set$ 
  assumes strict-mono-sets  $A\ f$ 
  shows pairwise  $(\lambda x\ y. disjoint\ (f\ x)\ (f\ y))\ A$ 
  using assms unfolding strict-mono-sets-def pairwise-def
  by (meson antisym-conv3 disjnt-sym less-sets-imp-disjnt)

```

**lemma** *strict-mono-sets-subset*:

**assumes** *strict-mono-sets*  $B f A \subseteq B$

**shows** *strict-mono-sets*  $A f$

**using** *assms* **by** (*auto simp: strict-mono-sets-def*)

**lemma** *strict-mono-less-sets-Min*:

**assumes** *strict-mono-sets*  $I f$  *finite*  $I I \neq \{\}$

**shows** *less-sets*  $(f (Min I)) (\cup (f ' (I - \{Min I\})))$

**using** *assms* **by** (*simp add: strict-mono-sets-def less-sets-UN2 dual-order.strict-iff-order*)

**lemma** *pair-less-iff1* [*simp*]:  $((x,y), (x,z)) \in \text{pair-less} \longleftrightarrow y < z$

**by** (*simp add: pair-less-def*)

**lemma** *infinite-finite-Inter*:

**assumes** *finite*  $\mathcal{A} \mathcal{A} \neq \{\}$   $\bigwedge A. A \in \mathcal{A} \implies \text{infinite } A$

**and**  $\bigwedge A B. \llbracket A \in \mathcal{A}; B \in \mathcal{A} \rrbracket \implies A \cap B \in \mathcal{A}$

**shows** *infinite*  $(\bigcap \mathcal{A})$

**by** (*simp add: assms finite-Inf-in*)

**lemma** *atLeast-less-sets*:  $\llbracket \text{less-sets } A \{x\}; B \subseteq \{x..\} \rrbracket \implies \text{less-sets } A B$

**by** (*force simp: less-sets-def subset-iff*)

## 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*  $(\text{insert } a A) = a \# \text{sorted-list-of-set } A$

**proof** –

**have** *strict-sorted*  $(a \# \text{sorted-list-of-set } A)$

**using** *assms less-setsD* **by** *auto*

**moreover** **have** *list.set*  $(a \# \text{sorted-list-of-set } A) = \text{insert } a A$

**using** *assms* **by** *force*

**moreover** **have** *length*  $(a \# \text{sorted-list-of-set } A) = \text{card } (\text{insert } a A)$

**using** *assms card-insert-if less-setsD* **by** *fastforce*

**ultimately** **show** *?thesis*

**by** (*metis*  $\langle \text{finite } A \rangle$  *finite-insert sorted-list-of-set-unique*)

**qed**

**lemma** *sorted-list-of-set-Un*:

**assumes**  $AB: \text{less-sets } A B$  **and** *fin*: *finite*  $A$  *finite*  $B$

**shows** *sorted-list-of-set*  $(A \cup B) = \text{sorted-list-of-set } A @ \text{sorted-list-of-set } B$

**proof** –

**have** *strict-sorted*  $(\text{sorted-list-of-set } A @ \text{sorted-list-of-set } B)$

**using**  $AB$  **unfolding** *less-sets-def*

**by** (*metis* *fin* *set-sorted-list-of-set sorted-wrt-append strict-sorted-list-of-set*)

**moreover** **have** *card*  $A + \text{card } B = \text{card } (A \cup B)$

**using** *less-sets-imp-disjnt* [ $OF AB$ ]

**by** (*simp add: assms card-Un-disjoint disjnt-def*)

**ultimately** **show** *?thesis*

by (*simp add: assms strict-sorted-equal*)  
 qed

**lemma** *sorted-list-of-set-UN-lessThan*:

fixes  $k::nat$   
 assumes  $sm: strict\ mono\ sets \{..<k\} A$  and  $\bigwedge i. i < k \implies finite (A i)$   
 shows  $sorted\ list\ of\ set (\bigcup_{i<k}. A i) = concat (map (sorted\ list\ of\ set \circ A) (sorted\ list\ of\ set \{..<k\}))$   
 using *assms*  
**proof** (*induction k*)  
 case 0  
 then show ?*case*  
 by *auto*  
 next  
 case (*Suc k*)  
 have  $ls: less\ sets (\bigcup (A ' \{..<k\})) (A k)$   
 using  $sm\ Suc.prem\ 1$  *strict-mono-setsD* **by** (*force simp: less-sets-UN1*)  
 have  $sorted\ list\ of\ set (\bigcup (A ' \{..<Suc\ k\})) = sorted\ list\ of\ set (\bigcup (A ' \{..<k\}) \cup A k)$   
 by (*simp add: Un-commute lessThan-Suc*)  
 also have  $\dots = sorted\ list\ of\ set (\bigcup (A ' \{..<k\})) @ sorted\ list\ of\ set (A k)$   
 by (*rule sorted-list-of-set-Un*) (*auto simp: Suc.prem\ ls*)  
 also have  $\dots = concat (map (sorted\ list\ of\ set \circ A) (sorted\ list\ of\ set \{..<k\})) @ sorted\ list\ of\ set (A k)$   
 using *Suc strict-mono-sets-def* **by** *fastforce*  
 also have  $\dots = concat (map (sorted\ list\ of\ set \circ A) (sorted\ list\ of\ set \{..<Suc\ k\}))$   
 using *strict-mono-sets-def* **by** *fastforce*  
 finally show ?*case* .  
 qed

**lemma** *sorted-list-of-set-UN-atMost*:

fixes  $k::nat$   
 assumes  $strict\ mono\ sets \{..k\} A$  and  $\bigwedge i. i \leq k \implies finite (A i)$   
 shows  $sorted\ list\ of\ set (\bigcup_{i\leq k}. A i) = concat (map (sorted\ list\ of\ set \circ A) (sorted\ list\ of\ set \{..k\}))$   
 by (*metis assms lessThan-Suc-atMost less-Suc-eq-le sorted-list-of-set-UN-lessThan*)

### 1.3 Monotonic enumeration of a countably infinite set

**abbreviation**  $enum \equiv enumerate$

Could be generalised to infinite countable sets of any type

**lemma** *nat-infinite-iff*:

fixes  $N :: nat\ set$   
 shows  $infinite\ N \iff (\exists f::nat \Rightarrow nat. N = range\ f \wedge strict\ mono\ f)$   
**proof** *safe*  
 assume  $infinite\ N$   
 then show  $\exists f. N = range (f::nat \Rightarrow nat) \wedge strict\ mono\ f$   
 by (*metis bij-betw-imp-surj-on bij-enumerate enumerate-mono strict-mono-def*)

```

next
  fix  $f :: \text{nat} \Rightarrow \text{nat}$ 
  assume  $\text{strict-mono } f$  and  $N = \text{range } f$  and  $\text{finite } (\text{range } f)$ 
  then show  $\text{False}$ 
    using  $\text{range-inj-infinite strict-mono-imp-inj-on}$  by  $\text{blast}$ 
qed

lemma  $\text{enum-works}$ :
  fixes  $N :: \text{nat set}$ 
  assumes  $\text{infinite } N$ 
  shows  $N = \text{range } (\text{enum } N) \wedge \text{strict-mono } (\text{enum } N)$ 
  by  $(\text{metis } \text{assms } \text{bij-betw-imp-surj-on } \text{bij-enumerate } \text{enumerate-mono } \text{strict-monoI})$ 

lemma  $\text{range-enum}$ :  $\text{range } (\text{enum } N) = N$  and  $\text{strict-mono-enum}$ :  $\text{strict-mono } (\text{enum } N)$ 
  if  $\text{infinite } N$  for  $N :: \text{nat set}$ 
  using  $\text{enum-works [OF that]}$  by  $\text{auto}$ 

lemma  $\text{enum-0-eq-Inf}$ :
  fixes  $N :: \text{nat set}$ 
  assumes  $\text{infinite } N$ 
  shows  $\text{enum } N \ 0 = \text{Inf } N$ 
proof –
  have  $\text{enum } N \ 0 \in N$ 
    using  $\text{assms } \text{range-enum}$  by  $\text{auto}$ 
  moreover have  $\bigwedge x. x \in N \implies \text{enum } N \ 0 \leq x$ 
    by  $(\text{metis } (\text{mono-tags, } \text{opaque-lifting}) \text{ assms } \text{imageE } \text{le0 } \text{less-mono-imp-le-mono } \text{range-enum } \text{strict-monoD } \text{strict-mono-enum})$ 
  ultimately show  $?thesis$ 
    by  $(\text{metis } \text{cInf-eq-minimum})$ 
qed

lemma  $\text{enum-works-finite}$ :
  fixes  $N :: \text{nat set}$ 
  assumes  $\text{finite } N$ 
  shows  $N = \text{enum } N \ ' \ \{..<\text{card } N\} \wedge \text{strict-mono-on } \{..<\text{card } N\} (\text{enum } N)$ 
  using  $\text{assms}$ 
  by  $(\text{metis } \text{bij-betw-imp-surj-on } \text{finite-bij-enumerate } \text{finite-enumerate-mono } \text{lessThan-iff } \text{strict-mono-onI})$ 

lemma  $\text{enum-obtain-index-finite}$ :
  fixes  $N :: \text{nat set}$ 
  assumes  $x \in N$   $\text{finite } N$ 
  obtains  $i$  where  $i < \text{card } N$   $x = \text{enum } N \ i$ 
  by  $(\text{metis } \langle x \in N \rangle \langle \text{finite } N \rangle \text{enum-works-finite } \text{imageE } \text{lessThan-iff})$ 

lemma  $\text{enum-0-eq-Inf-finite}$ :
  fixes  $N :: \text{nat set}$ 
  assumes  $\text{finite } N$   $N \neq \{\}$ 

```

**shows**  $\text{enum } N \ 0 = \text{Inf } N$   
**proof** –  
**have**  $\text{enum } N \ 0 \in N$   
**by** (*metis* *Nat.neq0-conv* *assms* *empty-is-image* *enum-works-finite* *image-eqI* *lessThan-empty-iff* *lessThan-iff*)  
**moreover have**  $\text{enum } N \ 0 \leq x$  **if**  $x \in N$  **for**  $x$   
**proof** –  
**obtain**  $i$  **where**  $i < \text{card } N$   $x = \text{enum } N \ i$   
**by** (*metis*  $\langle x \in N \rangle$   $\langle \text{finite } N \rangle$  *enum-obtain-index-finite*)  
**with** *assms* **show** *?thesis*  
**by** (*metis* *Nat.neq0-conv* *finite-enumerate-mono* *less-or-eq-imp-le*)  
**qed**  
**ultimately show** *?thesis*  
**by** (*metis* *cInf-eq-minimum*)  
**qed**

**lemma** *greaterThan-less-enum*:  
**fixes**  $N :: \text{nat set}$   
**assumes**  $N \subseteq \{x < ..\}$  *infinite*  $N$   
**shows**  $x < \text{enum } N \ i$   
**using** *assms* *range-enum* **by** *fastforce*

**lemma** *atLeast-le-enum*:  
**fixes**  $N :: \text{nat set}$   
**assumes**  $N \subseteq \{x ..\}$  *infinite*  $N$   
**shows**  $x \leq \text{enum } N \ i$   
**using** *assms* *range-enum* **by** *fastforce*

**lemma** *less-sets-empty1* [*simp*]: *less-sets*  $\{\}$   $A$  **and** *less-sets-empty2* [*simp*]: *less-sets*  $A \ \{\}$   
**by** (*simp-all* *add*: *less-sets-def*)

**lemma** *less-sets-singleton1* [*simp*]: *less-sets*  $\{a\}$   $A \longleftrightarrow (\forall x \in A. a < x)$   
**and** *less-sets-singleton2* [*simp*]: *less-sets*  $A \ \{a\} \longleftrightarrow (\forall x \in A. x < a)$   
**by** (*simp-all* *add*: *less-sets-def*)

**lemma** *less-sets-atMost* [*simp*]: *less-sets*  $\{..a\}$   $A \longleftrightarrow (\forall x \in A. a < x)$   
**and** *less-sets-atLeast* [*simp*]: *less-sets*  $A \ \{a.. \}$   $\longleftrightarrow (\forall x \in A. x < a)$   
**by** (*auto simp*: *less-sets-def*)

**lemma** *less-sets-imp-strict-mono-sets*:  
**assumes**  $\bigwedge i. \text{less-sets } (A \ i) \ (A \ (\text{Suc } i)) \ \bigwedge i. i > 0 \implies A \ i \neq \{\}$   
**shows** *strict-mono-sets* *UNIV*  $A$   
**proof** (*clarsimp simp*: *strict-mono-sets-def*)  
**fix**  $i \ j :: \text{nat}$   
**assume**  $i < j$   
**then show** *less-sets*  $(A \ i) \ (A \ j)$   
**proof** (*induction*  $j - i$  *arbitrary*:  $i \ j$ )  
**case** (*Suc*  $x$ )



```

    then show ?case
      by (metis Suc-diff-Suc Suc-inject Suc-mono assms less-Suc-eq less-sets-trans
zero-less-Suc)
    qed auto
  qed

```

```

lemma less-sets-Suc-Max:
  assumes finite A
  shows less-sets A {Suc (Max A)..}
proof (cases A = {})
  case False
  then show ?thesis
    by (simp add: assms less-Suc-eq-le)
qed auto

```

```

lemma infinite-nat-greaterThan:
  fixes m::nat
  assumes infinite N
  shows infinite (N ∩ {m<..})
proof -
  have N ⊆ -{m<..} ∪ (N ∩ {m<..})
    by blast
  moreover have finite (-{m<..})
    by simp
  ultimately show ?thesis
    using assms finite-subset by blast
qed

```

end

## 2 Ordinal Partitions

Material from Jean A. Larson, A short proof of a partition theorem for the ordinal  $\omega^\omega$ . *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-1st :: [('a × 'a) set, 'a set, V list, nat] ⇒ bool
  where partn-1st r B α n ≡ ∀ f ∈ [B]n → {..length α}.

```

$\exists i < \text{length } \alpha. \exists H. H \subseteq B \wedge \text{ordertype } H r = (\alpha!i) \wedge f' (nsets H n)$   
 $\subseteq \{i\}$

**abbreviation** *partn-lst-VWF* ::  $V \Rightarrow V \text{ list} \Rightarrow \text{nat} \Rightarrow \text{bool}$   
**where** *partn-lst-VWF*  $\beta \equiv \text{partn-lst VWF } (\text{elts } \beta)$

**lemma** *partn-lst-E*:

**assumes** *partn-lst*  $r B \alpha n f \in nsets B n \rightarrow \{..<l\} l = \text{length } \alpha$   
**obtains**  $i H$  **where**  $i < l H \subseteq B$   
 $\text{ordertype } H r = \alpha!i f' (nsets H n) \subseteq \{i\}$   
**using** *assms* **by** (*auto simp: partn-lst-def*)

**lemma** *partn-lst-VWF-nontriv*:

**assumes** *partn-lst-VWF*  $\beta \alpha n l = \text{length } \alpha \text{ Ord } \beta l > 0$   
**obtains**  $i$  **where**  $i < l \alpha!i \leq \beta$

**proof** –

**have**  $\{..<l\} \neq \{\}$   
**by** (*simp add: <l > 0 lessThan-empty-iff*)  
**then obtain**  $f$  **where**  $f \in nsets (\text{elts } \beta) n \rightarrow \{..<l\}$   
**by** (*meson Pi-eq-empty equals0I*)  
**then obtain**  $i H$  **where**  $i < l H \subseteq \text{elts } \beta$  **and** *eq: tp*  $H = \alpha!i$   
**using** *assms* **by** (*metis partn-lst-E*)  
**then have**  $\alpha!i \leq \beta$   
**by** (*metis <H ⊆ elts β> <Ord β> eq ordertype-le-Ord*)  
**then show** *thesis*  
**using**  $\langle i < l \rangle$  **that** **by** *auto*

**qed**

**lemma** *partn-lst-triv0*:

**assumes**  $\alpha!i = 0 i < \text{length } \alpha n \neq 0$   
**shows** *partn-lst*  $r B \alpha n$   
**by** (*metis partn-lst-def assms bot-least image-empty nsets-empty-iff ordertype-empty*)

**lemma** *partn-lst-triv1*:

**assumes**  $\alpha!i \leq 1 i < \text{length } \alpha n > 1 B \neq \{\}$  *wf*  $r$   
**shows** *partn-lst*  $r B \alpha n$   
**unfolding** *partn-lst-def*

**proof** *clarsimp*

**obtain**  $\gamma$  **where**  $\gamma \in B \alpha \neq []$   
**using** *assms mem-0-Ord* **by** *fastforce*  
**have** *01*:  $\alpha!i = 0 \vee \alpha!i = 1$   
**using** *assms* **by** (*fastforce simp: one-V-def*)  
**fix**  $f$   
**assume**  $f: f \in [B]^n \rightarrow \{..<\text{length } \alpha\}$   
**with** *assms* **have**  $\text{ordertype } \{\gamma\} r = 1 \wedge f' [\{\gamma\}]^n \subseteq \{i\}$   
 $\text{ordertype } \{\} r = 0 \wedge f' [\{\} ]^n \subseteq \{i\}$   
**by** (*auto simp: one-V-def ordertype-insert nsets-eq-empty*)  
**with** *assms 01* **show**  $\exists i < \text{length } \alpha. \exists H \subseteq B. \text{ordertype } H r = \alpha!i \wedge f' [H]^n \subseteq \{i\}$

using  $\langle \gamma \in B \rangle$  by auto  
qed

**lemma** *partn-lst-two-swap*:

assumes *partn-lst*  $r$   $B$   $[x,y]$   $n$  shows *partn-lst*  $r$   $B$   $[y,x]$   $n$

**proof** –

{ **fix**  $f :: 'a \text{ set} \Rightarrow \text{nat}$   
**assume**  $f: f \in [B]^n \rightarrow \{..<2\}$   
**then have**  $f': (\lambda i. 1 - i) \circ f \in [B]^n \rightarrow \{..<2\}$   
**by** (*auto simp: Pi-def*)  
**obtain**  $i H$  **where**  $i < 2$   $H \subseteq B$  *ordertype*  $H$   $r = ([x,y]!i) ((\lambda i. 1 - i) \circ f)'$   
(*nsets*  $H$   $n$ )  $\subseteq \{i\}$   
**by** (*auto intro: partn-lst-E [OF assms f']*)  
**moreover have**  $f x = \text{Suc } 0$  **if**  $\text{Suc } 0 \leq f x$   $x \in [H]^n$  **for**  $x$   
**using**  $f$  *that*  $\langle H \subseteq B \rangle$  *nsets-mono* **by** (*fastforce simp: Pi-iff*)  
**ultimately have** *ordertype*  $H$   $r = [y,x] ! (1-i) \wedge f' [H]^n \subseteq \{1-i\}$   
**by** (*force simp: eval-nat-numeral less-Suc-eq*)  
**then have**  $\exists i H. i < 2 \wedge H \subseteq B \wedge$  *ordertype*  $H$   $r = [y,x] ! i \wedge f' [H]^n \subseteq \{i\}$   
**by** (*metis Suc-1*  $\langle H \subseteq B \rangle$  *diff-less-Suc*) }  
**then show** *?thesis*  
**by** (*auto simp: partn-lst-def eval-nat-numeral*)  
qed

**lemma** *partn-lst-greater-resource*:

assumes  $M: \text{partn-lst } r$   $B$   $\alpha$   $n$  **and**  $B \subseteq C$

shows *partn-lst*  $r$   $C$   $\alpha$   $n$

**proof** (*clarsimp simp: partn-lst-def*)

**fix**  $f$

**assume**  $f \in [C]^n \rightarrow \{..<\text{length } \alpha\}$

**then have**  $f \in [B]^n \rightarrow \{..<\text{length } \alpha\}$

**by** (*metis*  $\langle B \subseteq C \rangle$  *part-fn-def part-fn-subset*)

**then obtain**  $i H$  **where**  $i < \text{length } \alpha$

**and**  $H \subseteq B$  *ordertype*  $H$   $r = (\alpha!i)$

**and**  $f' \text{ nsets } H$   $n \subseteq \{i\}$

**using**  $M$  *partn-lst-def* **by** *metis*

**then show**  $\exists i < \text{length } \alpha. \exists H \subseteq C. \text{ordertype } H$   $r = \alpha ! i \wedge f' [H]^n \subseteq \{i\}$

**using**  $\langle B \subseteq C \rangle$  **by** *blast*

qed

**lemma** *partn-lst-less*:

assumes  $M: \text{partn-lst } r$   $B$   $\alpha$   $n$  **and**  $\text{eq: length } \alpha' = \text{length } \alpha$  **and**  $\text{List.set } \alpha' \subseteq \text{ON}$

**and**  $le: \bigwedge i. i < \text{length } \alpha \implies \alpha'!i \leq \alpha!i$

**and**  $r: \text{wf } r$  *trans*  $r$  *total-on*  $B$   $r$  **and** *small*  $B$

shows *partn-lst*  $r$   $B$   $\alpha'$   $n$

**proof** (*clarsimp simp: partn-lst-def*)

**fix**  $f$

**assume**  $f \in [B]^n \rightarrow \{..<\text{length } \alpha'\}$

**then obtain**  $i \in H$  **where**  $i < \text{length } \alpha$   
**and**  $H \subseteq B$  **small**  $H$  **and**  $H: \text{ordertype } H \ r = (\alpha!i)$   
**and**  $f: \text{‘ } n\text{-sets } H \ n \subseteq \{i\}$   
**using** *assms* **by** (*auto simp: partn-lst-def smaller-than-small*)  
**then have**  $\text{bij: } \text{bij-betw } (\text{ordermap } H \ r) \ H \ (\text{elts } (\alpha!i))$   
**using** *ordermap-bij [of r H] r*  
**by** (*smt <small B> in-mono smaller-than-small total-on-def*)  
**define**  $H'$  **where**  $H' = \text{inv-into } H \ (\text{ordermap } H \ r) \ \text{‘ } (\text{elts } (\alpha!i))$   
**have**  $H' \subseteq H$   
**using**  $\text{bij } \langle i < \text{length } \alpha \rangle \text{ bij-betw-imp-surj-on } le$   
**by** (*force simp: H'-def image-subset-iff intro: inv-into-into*)  
**moreover have**  $ot: \text{ordertype } H' \ r = (\alpha!i)$   
**proof** (*subst ordertype-eq-iff*)  
**show**  $\text{Ord } (\alpha!i)$   
**using** *assms* **by** (*simp add: <i < length alpha> subset-eq*)  
**show** *small H'*  
**by** (*simp add: H'-def*)  
**show**  $\exists f. \text{bij-betw } f \ H' \ (\text{elts } (\alpha!i)) \wedge (\forall x \in H'. \forall y \in H'. (f \ x < f \ y) = ((x, y) \in r))$   
**proof** (*intro exI conjI ballI*)  
**show**  $\text{bij-betw } (\text{ordermap } H \ r) \ H' \ (\text{elts } (\alpha!i))$   
**using**  $\langle H' \subseteq H \rangle$   
**by** (*metis H'-def <i < length alpha> bij bij-betw-inv-into-RIGHT bij-betw-subset le less-eq-V-def*)  
**show**  $(\text{ordermap } H \ r \ x < \text{ordermap } H \ r \ y) = ((x, y) \in r)$   
**if**  $x \in H' \ y \in H'$  **for**  $x \ y$   
**proof** (*intro iffI ordermap-mono-less*)  
**assume**  $\text{ordermap } H \ r \ x < \text{ordermap } H \ r \ y$   
**then show**  $(x, y) \in r$   
**by** (*metis <H subseteq B> <small H> <H' subseteq H> leD ordermap-mono-le r subsetD that total-on-def*)  
**qed** (*use assms that <H' subseteq H> <small H> in auto*)  
**qed**  
**show** *total-on H' r*  
**using**  $r$  **by** (*meson <H subseteq B> <H' subseteq H> subsetD total-on-def*)  
**qed** (*use r in auto*)  
**ultimately show**  $\exists i < \text{length } \alpha'. \exists H \subseteq B. \text{ordertype } H \ r = \alpha!i \wedge f \ \text{‘ } [H]^n \subseteq \{i\}$   
**using**  $\langle H \subseteq B \rangle \langle i < \text{length } \alpha \rangle f$  *assms*  
**by** (*metis image-mono nsets-mono subset-trans*)  
**qed**

Holds because no  $n$ -sets exist!

**lemma** *partn-lst-VWF-degenerate*:

**assumes**  $k < n$

**shows** *partn-lst-VWF*  $\omega \ (\text{ord-of-nat } k \ \# \ \alpha \ s) \ n$

**proof** (*clarsimp simp: partn-lst-def*)

**fix**  $f :: V \ \text{set} \Rightarrow \text{nat}$

**have**  $[\text{elts } (\text{ord-of-nat } k)]^n = \{\}$

**by** (*simp add: nsets-eq-empty assms finite-Ord-omega*)

**then have**  $f \text{ ' } [elts \text{ (ord-of-nat } k)]^n \subseteq \{0\}$   
**by** *auto*  
**then show**  $\exists i < Suc \text{ (length } \alpha s). \exists H \subseteq elts \ \omega. tp \ H = \text{ (ord-of-nat } k \ \# \ \alpha s) ! \ i \wedge$   
 $f \text{ ' } [H]^n \subseteq \{i\}$   
**using** *assms ordertype-eq-Ord [of ord-of-nat k] elts-ord-of-nat less-Suc-eq-0-disj*  
**by** *fastforce*  
**qed**

**lemma** *partn-lst-VWF- $\omega$ -2:*

**assumes** *Ord*  $\alpha$

**shows** *partn-lst-VWF*  $(\omega \uparrow (1+\alpha)) [2, \omega \uparrow (1+\alpha)] \ 2$  (**is** *partn-lst-VWF*  $? \beta$  - -)

**proof** (*clarsimp simp: partn-lst-def*)

**fix**  $f$

**assume**  $f: f \in [elts \ ?\beta]^2 \rightarrow \{..<Suc \text{ (Suc } 0)\}$

**show**  $\exists i < Suc \text{ (Suc } 0). \exists H \subseteq elts \ ?\beta. tp \ H = [2, \ ?\beta] ! \ i \wedge f \text{ ' } [H]^2 \subseteq \{i\}$

**proof** (*cases*  $\exists x \in elts \ ?\beta. \exists y \in elts \ ?\beta. x \neq y \wedge f\{x,y\} = 0$ )

**case** *True*

**then obtain**  $x \ y$  **where**  $x \in elts \ ?\beta \ y \in elts \ ?\beta \ x \neq y \ f \{x, y\} = 0$

**by** *auto*

**then have**  $\{x,y\} \subseteq elts \ ?\beta \ tp \ \{x,y\} = 2 \ f \text{ ' } [\{x, y\}]^2 \subseteq \{0\}$

**by** *auto (simp add: eval-nat-numeral ordertype-VWF-finite-nat)*

**with**  $\langle x \neq y \rangle$  **show** *?thesis*

**by** (*metis nth-Cons-0 zero-less-Suc*)

**next**

**case** *False*

**with**  $f$  **have**  $\forall x \in elts \ ?\beta. \forall y \in elts \ ?\beta. x \neq y \longrightarrow f \{x, y\} = 1$

**unfolding** *Pi-iff* **using** *lessThan-Suc* **by** *force*

**then have**  $tp \ (elts \ ?\beta) = ?\beta \ f \text{ ' } [elts \ ?\beta]^2 \subseteq \{Suc \ 0\}$

**by** (*auto simp: assms nsets-2-eq*)

**then show** *?thesis*

**by** (*metis lessI nth-Cons-0 nth-Cons-Suc subsetI*)

**qed**

**qed**

## 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*  $r \ U \ \alpha \ n$  **and**  $\beta: ordertype \ U \ r = \beta$  **and** *small*  $U$

**and**  $r: wf \ r \ trans \ r \ total-on \ U \ r$

**shows** *partn-lst-VWF*  $\beta \ \alpha \ n$

**unfolding** *partn-lst-def*

**proof** *clarsimp*

**fix**  $f$

**assume**  $f: f \in [elts \ \beta]^n \rightarrow \{..<length \ \alpha\}$

**define**  $cv$  **where**  $cv \equiv \lambda X. ordermap \ U \ r \text{ ' } X$

**have**  $bij: bij\text{-betw} \ (ordermap \ U \ r) \ U \ (elts \ \beta)$

**using** *ordermap-bij [of r U] assms* **by** *blast*

**then have**  $bij\text{-}cv: bij\text{-betw} \ cv \ ([U]^n) \ ([elts \ \beta]^n)$

```

    using bij-betw-nsets cv-def by blast
  then have func:  $f \circ cv \in [U]^n \rightarrow \{..<length\ \alpha\}$  and inj-on (ordermap  $U\ r$ )  $U$ 
    using bij bij-betw-def bij-betw-apply f by fastforce+
  then have cv-part:  $\llbracket \forall x \in [X]^n. f\ (cv\ x) = i; X \subseteq U; a \in [cv\ X]^n \rrbracket \implies f\ a = i$ 
for a X i n
  by (force simp: cv-def nsets-def subset-image-iff inj-on-subset finite-image-iff card-image)
  have ot-eq [simp]:  $tp\ (cv\ X) = ordertype\ X\ r$  if  $X \subseteq U$  for  $X$ 
    unfolding cv-def
    by (meson <small U> ordertype-image-ordermap r that total-on-subset)
  obtain  $X\ i$  where  $X \subseteq U$  and  $X$ :  $ordertype\ X\ r = \alpha!i$  ( $f \circ cv$ ) ' $[X]^n \subseteq \{i\}$ '
    and  $i < length\ \alpha$ 
    using part func by (auto simp: partn-lst-def)
  show  $\exists i < length\ \alpha. \exists H \subseteq elts\ \beta. tp\ H = \alpha!i \wedge f\ '[H]^n \subseteq \{i\}$ 
  proof (intro exI conjI)
    show  $i < length\ \alpha$ 
      by (simp add: <i < length alpha>)
    show  $cv\ X \subseteq elts\ \beta$ 
      using  $\langle X \subseteq U \rangle$  bij bij-betw-imp-surj-on cv-def by blast
    show  $tp\ (cv\ X) = \alpha!i$ 
      by (simp add: X(1) <X subset U>)
    show  $f\ '[cv\ X]^n \subseteq \{i\}$ 
      using  $X\ \langle X \subseteq U \rangle$  cv-part unfolding image-subset-iff cv-def
      by (metis comp-apply insertCI singletonD)
  qed
qed

```

```

lemma partn-lst-imp-partn-lst-VWF:
  assumes part: partn-lst  $r\ U\ \alpha\ n$  and  $\beta$ :  $ordertype\ U\ r \leq \beta$  small U
    and  $r$ : wf r trans r total-on U r
  shows partn-lst-VWF  $\beta\ \alpha\ n$ 
  by (metis assms less-eq-V-def partn-lst-imp-partn-lst-VWF-eq partn-lst-greater-resource)

```

```

lemma partn-lst-VWF-imp-partn-lst-eq:
  assumes part: partn-lst-VWF  $\beta\ \alpha\ n$  and  $\beta$ :  $ordertype\ U\ r = \beta$  small U
    and  $r$ : wf r trans r total-on U r
  shows partn-lst  $r\ U\ \alpha\ n$ 
  unfolding partn-lst-def
proof clarsimp
  fix  $f$ 
  assume  $f$ :  $f \in [U]^n \rightarrow \{..<length\ \alpha\}$ 
  define cv where  $cv \equiv \lambda X. inv-into\ U\ (ordermap\ U\ r)\ '[X$ 
  have bij: bij-betw (ordermap  $U\ r$ )  $U$  ( $elts\ \beta$ )
    using ordermap-bij [of r U] assms by blast
  then have bij-cv: bij-betw  $cv\ ([elts\ \beta]^n)\ ([U]^n)$ 
    using bij-betw-nsets bij-betw-inv-into unfolding cv-def by blast
  then have func:  $f \circ cv \in [elts\ \beta]^n \rightarrow \{..<length\ \alpha\}$ 
    using bij-betw-apply f by fastforce
  have inj-on (ordermap  $U\ r$ )  $U$ 

```

**using** *bij bij-betw-def* **by** *blast*  
**then have** *cv-part*:  $\llbracket \forall x \in [X]^n. f (cv\ x) = i; X \subseteq \text{elts } \beta; a \in [cv\ X]^n \rrbracket \implies f\ a =$   
*i* **for** *a X i n*  
**by** (*metis bij bij-betw-imp-surj-on cv-def inj-on-inv-into nset-image-obtains*)  
**have** *ot-eq [simp]*: *ordertype (cv X) r = tp X* **if**  $X \subseteq \text{elts } \beta$  **for** *X*  
**unfolding** *cv-def*  
**proof** (*rule ordertype-inc-eq*)  
**show** *small X*  
**using** *down that by auto*  
**show** (*inv-into U (ordermap U r) x, inv-into U (ordermap U r) y*)  $\in r$   
**if**  $x \in X\ y \in X$  **and**  $(x,y) \in VWF$  **for**  $x\ y$   
**proof** –  
**have** *xy*:  $x \in \text{ordermap } U\ r\ \text{‘ } U\ y \in \text{ordermap } U\ r\ \text{‘ } U$   
**using**  $\langle X \subseteq \text{elts } \beta \rangle\ \langle x \in X \rangle\ \langle y \in X \rangle$  *bij bij-betw-imp-surj-on* **by** *blast+*  
**then have** *eq*:  $\text{ordermap } U\ r\ (\text{inv-into } U\ (\text{ordermap } U\ r)\ x) = x\ \text{ordermap } U$   
 $r\ (\text{inv-into } U\ (\text{ordermap } U\ r)\ y) = y$   
**by** (*meson f-inv-into-f*)  
**then have**  $y \notin \text{elts } x$   
**by** (*metis (no-types) VWF-non-refl mem-imp-VWF that(3) trans-VWF*  
*trans-def*)  
**with** *r* **show** *?thesis*  
**by** (*metis (no-types) VWF-non-refl xy eq assms(3) inv-into-into ordermap-mono*  
*that(3) total-on-def*)  
**qed**  
**qed** (*use r in auto*)  
**obtain** *X i* **where**  $X \subseteq \text{elts } \beta$  **and**  $X: tp\ X = \alpha!i\ (f \circ cv)\ \text{‘ } [X]^n \subseteq \{i\}$   
**and**  $i < \text{length } \alpha$   
**using** *part func* **by** (*auto simp: partn-lst-def*)  
**show**  $\exists i < \text{length } \alpha. \exists H \subseteq U. \text{ordertype } H\ r = \alpha!i \wedge f\ \text{‘ } [H]^n \subseteq \{i\}$   
**proof** (*intro exI conjI*)  
**show**  $i < \text{length } \alpha$   
**by** (*simp add: <i < length alpha>*)  
**show**  $cv\ X \subseteq U$   
**using**  $\langle X \subseteq \text{elts } \beta \rangle$  *bij bij-betw-imp-surj-on bij-betw-inv-into cv-def* **by** *blast*  
**show** *ordertype (cv X) r = alpha ! i*  
**by** (*simp add: X(1) <X subset elts beta>*)  
**show**  $f\ \text{‘ } [cv\ X]^n \subseteq \{i\}$   
**using**  $X\ \langle X \subseteq \text{elts } \beta \rangle$  *cv-part* **unfolding** *image-subset-iff cv-def*  
**by** (*metis comp-apply insertCI singletonD*)  
**qed**  
**qed**

**corollary** *partn-lst-VWF-imp-partn-lst*:  
**assumes** *partn-lst-VWF beta alpha n* **and**  $\beta: \text{ordertype } U\ r \geq \beta$  *small U*  
*wf r trans r total-on U r*  
**shows** *partn-lst r U alpha n*  
**by** (*metis assms less-eq-V-def partn-lst-VWF-imp-partn-lst-eq partn-lst-greater-resource*)

## 2.3 Simple consequences of the definitions

A restatement of the infinite Ramsey theorem using partition notation

**lemma** *Ramsey-partn: partn-lst-VWF*  $\omega$   $[\omega, \omega]$  2

**proof** (*clarsimp simp: partn-lst-def*)

**fix**  $f$

**assume**  $f \in [\text{elts } \omega]^2 \rightarrow \{..< \text{Suc } (\text{Suc } 0)\}$

**then have**  $*$ :  $\forall x \in \text{elts } \omega. \forall y \in \text{elts } \omega. x \neq y \longrightarrow f \{x, y\} < 2$

**by** (*auto simp: nsets-def eval-nat-numeral*)

**obtain**  $H$   $i$  **where**  $H: H \subseteq \text{elts } \omega$  **and** *infinite*  $H$

**and**  $t: i < \text{Suc } (\text{Suc } 0)$

**and**  $\text{teq}: \forall x \in H. \forall y \in H. x \neq y \longrightarrow f \{x, y\} = i$

**using** *Ramsey2 [OF infinite- $\omega$   $*$ ]* **by** (*auto simp: eval-nat-numeral*)

**then have**  $\text{tp } H = [\omega, \omega] ! i$

**using** *less-2-cases eval-nat-numeral ordertype-infinite- $\omega$*  **by force**

**moreover have**  $f' \{N. N \subseteq H \wedge \text{finite } N \wedge \text{card } N = 2\} \subseteq \{i\}$

**by** (*force simp: teq card-2-iff*)

**ultimately have**  $f' [H]^2 \subseteq \{i\}$

**by** (*metis (no-types) nsets-def numeral-2-eq-2*)

**then show**  $\exists i < \text{Suc } (\text{Suc } 0). \exists H \subseteq \text{elts } \omega. \text{tp } H = [\omega, \omega] ! i \wedge f' [H]^2 \subseteq \{i\}$

**using**  $H \langle \text{tp } H = [\omega, \omega] ! i \rangle t$  **by blast**

**qed**

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

**proposition** *omega-basic-counterexample:*

**assumes** *Ord*  $\alpha$

**shows**  $\neg \text{partn-lst-VWF } \alpha$   $[\text{succ } (\text{vcard } \alpha), \omega]$  2

**proof** –

**obtain**  $\pi$  **where**  $\text{fun } \pi: \pi \in \text{elts } \alpha \rightarrow \text{elts } (\text{vcard } \alpha)$  **and**  $\text{inj } \pi: \text{inj-on } \pi$   $(\text{elts } \alpha)$

**using** *inj-into-vcard* **by auto**

**have** *Ord*  $\pi$ : *Ord*  $(\pi x)$  **if**  $x \in \text{elts } \alpha$  **for**  $x$

**using** *Ord-in-Ord fun*  $\pi$  **that** **by fastforce**

**define**  $f$  **where**  $f A \equiv @i::\text{nat}. \exists x y. A = \{x, y\} \wedge x < y \wedge (\pi x < \pi y \wedge i=0 \vee \pi y < \pi x \wedge i=1)$  **for**  $A$

**have**  $f\text{-Pi}: f \in [\text{elts } \alpha]^2 \rightarrow \{..< \text{Suc } (\text{Suc } 0)\}$

**proof**

**fix**  $A$

**assume**  $A \in [\text{elts } \alpha]^2$

**then obtain**  $x y$  **where**  $xy: x \in \text{elts } \alpha \ y \in \text{elts } \alpha \ x < y$  **and**  $A: A = \{x, y\}$

**apply** (*clarsimp simp: nsets-2-eq*)

**by** (*metis Ord-in-Ord Ord-linear-lt assms insert-commute*)

**consider**  $\pi x < \pi y \mid \pi y < \pi x$

**by** (*metis Ord*  $\pi$  *Ord-linear-lt inj*  $\pi$  *inj-onD less-imp-not-eq2*  $xy$ )

**then show**  $f A \in \{..< \text{Suc } (\text{Suc } 0)\}$

**by** (*metis A One-nat-def lessI lessThan-iff zero-less-Suc*  $\langle x < y \rangle A \text{exE-some}$  [*OF - f-def*])

**qed**

**have**  $\text{fiff}: \pi x < \pi y \wedge i=0 \vee \pi y < \pi x \wedge i=1$

**if**  $f: f \{x, y\} = i$  **and**  $xy: x \in \text{elts } \alpha \ y \in \text{elts } \alpha \ x < y$  **for**  $x y i$



**proof** –  
**consider**  $\pi x < \pi y \mid \pi y < \pi x$   
**using**  $xy$  **by** (*metis Ord $\pi$  Ord-linear-lt inj $\pi$  inj-onD less-V-def*)  
**then show** *?thesis*  
**proof cases**  
**case 1**  
**then have**  $f\{x,y\} = 0$   
**using**  $\langle x < y \rangle$  **by** (*force simp: f-def Set.doubleton-eq-iff*)  
**then show** *?thesis*  
**using**  $1 f$  **by** *auto*  
**next**  
**case 2**  
**then have**  $f\{x,y\} = 1$   
**using**  $\langle x < y \rangle$  **by** (*force simp: f-def Set.doubleton-eq-iff*)  
**then show** *?thesis*  
**using**  $2 f$  **by** *auto*  
**qed**  
**qed**  
**have** *False*  
**if**  $eq: tp\ H = succ\ (vcard\ \alpha)$  **and**  $H: H \subseteq elts\ \alpha$   
**and**  $0: \bigwedge A. A \in [H]^2 \implies f\ A = 0$  **for**  $H$   
**proof** –  
**have** [*simp*]: *small H*  
**using**  $H$  **down by** *auto*  
**have**  $OH: Ord\ x$  **if**  $x \in H$  **for**  $x$   
**using**  $H$  *Ord-in-Ord*  $\langle Ord\ \alpha \rangle$  **that by** *blast*  
**have**  $\pi: \pi\ x < \pi\ y$  **if**  $x \in H\ y \in H\ x < y$  **for**  $x\ y$   
**using**  $0$  [*of*  $\{x,y\}$ ] *that H fiff* **by** (*fastforce simp: nsets-2-eq*)  
**have** *sub-vcad*:  $\pi\ ' H \subseteq elts\ (vcard\ \alpha)$   
**using**  $H\ fun\ \pi$  **by** *auto*  
**have**  $tp\ H = tp\ (\pi\ ' H)$   
**proof** (*rule ordertype-VWF-inc-eq [symmetric]*)  
**show**  $\pi\ ' H \subseteq ON$   
**using**  $H\ Ord\ \pi$  **by** *blast*  
**qed** (*auto simp:  $\pi\ OH\ subsetI$* )  
**also have**  $\dots \leq vcard\ \alpha$   
**by** (*simp add: H sub-vcad assms ordertype-le-Ord*)  
**finally show** *False*  
**by** (*simp add: eq succ-le-iff*)  
**qed**  
**moreover have** *False*  
**if**  $eq: tp\ H = \omega$  **and**  $H: H \subseteq elts\ \alpha$   
**and**  $1: \bigwedge A. A \in [H]^2 \implies f\ A = Suc\ 0$  **for**  $H$   
**proof** –  
**have** [*simp*]: *small H*  
**using**  $H$  **down by** *auto*  
**define**  $\eta$  **where**  $\eta \equiv inv\ into\ H\ (ordermap\ H\ VWF) \circ ord\ of\ nat$   
**have** *bij*: *bij-betw* (*ordermap H VWF*)  $H$  (*elts  $\omega$* )  
**by** (*metis ordermap-bij*  $\langle small\ H \rangle$  *eq total-on-VWF wf-VWF*)

**then have**  $\text{bij-betw } (\text{inv-into } H \ (\text{ordermap } H \ \text{VWF})) \ (\text{elts } \omega) \ H$   
**by** (*simp add: bij-betw-inv-into*)  
**then have**  $\eta: \text{bij-betw } \eta \ \text{UNIV } H$   
**unfolding**  $\eta\text{-def}$   
**by** (*metis  $\omega$ -def bij-betw-comp-iff2 bij-betw-def elts-of-set inf inj-ord-of-nat order-refl*)  
**moreover have**  $\text{Ord}\eta: \text{Ord } (\eta \ k) \ \text{for } k$   
**by** (*meson H Ord-in-Ord UNIV-I  $\eta$  assms bij-betw-apply subsetD*)  
**moreover obtain**  $k \ \text{where } k: (\pi \ (\eta(\text{Suc } k)), \pi \ (\eta \ k)) \notin \text{VWF}$   
**by** (*meson wf-VWF wf-iff-no-infinite-down-chain*)  
**moreover have**  $\pi: \pi \ y < \pi \ x \ \text{if } x \in H \ y \in H \ x < y \ \text{for } x \ y$   
**using** 1 [*of  $\{x,y\}$  that H fiff by (fastforce simp: nsets-2-eq)*]  
**ultimately have**  $\eta \ (\text{Suc } k) \leq \eta \ k$   
**by** (*metis H Ord $\pi$  Ord-linear2 VWF-iff-Ord-less bij-betw-def rangeI subset-iff*)  
**then have**  $(\eta \ (\text{Suc } k), \eta \ k) \in \text{VWF} \vee \eta \ (\text{Suc } k) = \eta \ k$   
**using**  $\text{Ord}\eta \ \text{Ord-mem-iff-lt}$  **by** *auto*  
**then have**  $\text{ordermap } H \ \text{VWF} \ (\eta \ (\text{Suc } k)) \leq \text{ordermap } H \ \text{VWF} \ (\eta \ k)$   
**by** (*metis  $\eta$   $\langle$ small  $H$  $\rangle$  bij-betw-imp-surj-on ordermap-mono-le rangeI trans-VWF wf-VWF*)  
**moreover have**  $\text{ordermap } H \ \text{VWF} \ (\eta \ (\text{Suc } k)) = \text{succ} \ (\text{ord-of-nat } k)$   
**unfolding**  $\eta\text{-def}$  **using** *bij bij-betw-inv-into-right* **by** *force*  
**moreover have**  $\text{ordermap } H \ \text{VWF} \ (\eta \ k) = \text{ord-of-nat } k$   
**using**  $\eta\text{-def}$  *bij bij-betw-inv-into-right* **by** *force*  
**ultimately show** *False*  
**by** (*simp add: less-eq-V-def*)  
**qed**  
**ultimately show** *?thesis*  
**apply** (*simp add: partn-lst-def image-subset-iff*)  
**by** (*metis f-Pi less-2-cases nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)  
**qed**

## 2.4 Specker's theorem

**definition** *form-split* ::  $[\text{nat}, \text{nat}, \text{nat}, \text{nat}, \text{nat}] \Rightarrow \text{bool}$  **where**  
 $\text{form-split } a \ b \ c \ d \ i \equiv a \leq c \wedge (i=0 \wedge a < b \wedge b < c \wedge c < d \vee$   
 $i=1 \wedge a < c \wedge c < b \wedge b < d \vee$   
 $i=2 \wedge a < c \wedge c < d \wedge d < b \vee$   
 $i=3 \wedge a=c \wedge b \neq d)$

**definition** *form* ::  $[(\text{nat} * \text{nat}) \text{set}, \text{nat}] \Rightarrow \text{bool}$  **where**  
 $\text{form } u \ i \equiv \exists a \ b \ c \ d. u = \{(a,b), (c,d)\} \wedge \text{form-split } a \ b \ c \ d \ i$

**definition** *scheme* ::  $[(\text{nat} * \text{nat}) \text{set}] \Rightarrow \text{nat set}$  **where**  
 $\text{scheme } u \equiv \text{fst } 'u \cup \text{snd } 'u$

**definition** *UU* ::  $(\text{nat} * \text{nat}) \ \text{set}$   
**where**  $UU \equiv \{(a,b). a < b\}$

**lemma** *ordertype-UNIV- $\omega$ 2*:  $\text{ordertype UNIV pair-less} = \omega \uparrow 2$

**using** *ordertype-Times* [of concl: UNIV UNIV less-than less-than]  
**by** (*simp add: total-less-than pair-less-def ordertype-nat- $\omega$  numeral-2-eq-2*)

**lemma** *ordertype-UU-ge- $\omega$ 2*: *ordertype UNIV pair-less  $\leq$  ordertype UU pair-less*  
**proof** (*rule ordertype-inc-le*)

**define**  $\pi$  **where**  $\pi \equiv \lambda(m,n). (m, \text{Suc } (m+n))$   
**show**  $(\pi (x::\text{nat} \times \text{nat}), \pi y) \in \text{pair-less}$  **if**  $(x, y) \in \text{pair-less}$  **for**  $x y$   
**using** *that by (auto simp:  $\pi$ -def pair-less-def split: prod.split)*  
**show**  $\text{range } \pi \subseteq \text{UU}$   
**by** (*auto simp:  $\pi$ -def UU-def*)

**qed** *auto*

**lemma** *ordertype-UU- $\omega$ 2*: *ordertype UU pair-less =  $\omega \uparrow 2$*

**by** (*metis eq-iff ordertype-UNIV- $\omega$ 2 ordertype-UU-ge- $\omega$ 2 ordertype-mono small top-greatest trans-pair-less wf-pair-less*)

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

**lemma** *lemma-2-3*:

**fixes**  $f :: (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{nat}$   
**assumes**  $f \in [\text{UU}]^2 \rightarrow \{..<\text{Suc } (\text{Suc } 0)\}$   
**obtains**  $N$  *js* **where** *infinite N and  $\bigwedge k u. [k < 4; u \in [\text{UU}]^2; \text{form } u k; \text{scheme } u \subseteq N] \implies f u = js!k$*

**proof** –

**have** *f-less2*:  $f \{p,q\} < \text{Suc } (\text{Suc } 0)$  **if**  $p \neq q$   $p \in \text{UU}$   $q \in \text{UU}$  **for**  $p q$

**proof** –

**have**  $\{p,q\} \in [\text{UU}]^2$   
**using** *that by (simp add: nsets-def)*  
**then show** *?thesis*  
**using** *assms by (simp add: Pi-iff)*

**qed**

**define**  $f0$  **where**  $f0 \equiv (\lambda A::\text{nat set. THE } x. \exists a b c d. A = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d \wedge x = f \{(a,b),(c,d)\})$

**have**  $f0$ :  $f0 \{a,b,c,d\} = f \{(a,b),(c,d)\}$  **if**  $a < b$   $b < c$   $c < d$  **for**  $a b c d$   
**unfolding** *f0-def*

**apply** (*rule theI2 [where a = f {(a,b), (c,d)}]*)  
**using** *that by (force simp: insert-eq-iff split: if-split-asm)+*

**have**  $f0 X < \text{Suc } (\text{Suc } 0)$   
**if** *finite X and card X = 4* **for**  $X$

**proof** –

**have**  $X \in [X]^4$   
**using** *that by (auto simp: nsets-def)*  
**then obtain**  $a b c d$  **where**  $X = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d$   
**by** (*auto simp: ordered-nsets-4-eq*)  
**then show** *?thesis*

**using**  $f0$  *f-less2* **by** (*auto simp: UU-def*)

**qed**

**then have**  $\exists N t. \text{infinite } N \wedge t < \text{Suc } (\text{Suc } 0)$

$\wedge (\forall X. X \subseteq N \wedge \text{finite } X \wedge \text{card } X = 4 \longrightarrow f0 X = t)$

**using** *Ramsey* [of *UNIV 4 f0 2*] **by** (*simp add: eval-nat-numeral*)  
**then obtain**  $N0\ j0$  **where** *infinite*  $N0$  **and**  $j0: j0 < \text{Suc}(\text{Suc } 0)$  **and**  $N0: \bigwedge A. A \in [N0]^4 \implies f0\ A = j0$   
**by** (*auto simp: nsets-def*)

**define**  $f1$  **where**  $f1 \equiv (\lambda A::\text{nat set. THE } x. \exists a\ b\ c\ d. A = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d \wedge x = f\ \{(a,c),(b,d)\})$   
**have**  $f1: f1\ \{a,b,c,d\} = f\ \{(a,c),(b,d)\}$  **if**  $a < b\ b < c\ c < d$  **for**  $a\ b\ c\ d$   
**unfolding** *f1-def*  
**apply** (*rule theI2 [where a = f\ \{(a,c), (b,d)\}]*)  
**using** *that by (force simp: insert-eq-iff split: if-split-asm)+*  
**have**  $f1\ X < \text{Suc}(\text{Suc } 0)$   
**if** *finite*  $X$  **and**  $\text{card } X = 4$  **for**  $X$   
**proof** –  
**have**  $X \in [X]^4$   
**using** *that by (auto simp: nsets-def)*  
**then obtain**  $a\ b\ c\ d$  **where**  $X = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d$   
**by** (*auto simp: ordered-nsets-4-eq*)  
**then show** *?thesis*  
**using**  $f1\ f\text{-less2}$  **by** (*auto simp: UU-def*)

**qed**

**then have**  $\exists N\ t. N \subseteq N0 \wedge \text{infinite } N \wedge t < \text{Suc}(\text{Suc } 0)$   
 $\wedge (\forall X. X \subseteq N \wedge \text{finite } X \wedge \text{card } X = 4 \implies f1\ X = t)$   
**using**  $\langle \text{infinite } N0 \rangle$  *Ramsey* [of *N0 4 f1 2*] **by** (*simp add: eval-nat-numeral*)  
**then obtain**  $N1\ j1$  **where**  $N1 \subseteq N0$  *infinite*  $N1$  **and**  $j1: j1 < \text{Suc}(\text{Suc } 0)$  **and**  
 $N1: \bigwedge A. A \in [N1]^4 \implies f1\ A = j1$   
**by** (*auto simp: nsets-def*)

**define**  $f2$  **where**  $f2 \equiv (\lambda A::\text{nat set. THE } x. \exists a\ b\ c\ d. A = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d \wedge x = f\ \{(a,d),(b,c)\})$   
**have**  $f2: f2\ \{a,b,c,d\} = f\ \{(a,d),(b,c)\}$  **if**  $a < b\ b < c\ c < d$  **for**  $a\ b\ c\ d$   
**unfolding** *f2-def*  
**apply** (*rule theI2 [where a = f\ \{(a,d), (b,c)\}]*)  
**using** *that by (force simp: insert-eq-iff split: if-split-asm)+*  
**have**  $f2\ X < \text{Suc}(\text{Suc } 0)$   
**if** *finite*  $X$  **and**  $\text{card } X = 4$  **for**  $X$   
**proof** –  
**have**  $X \in [X]^4$   
**using** *that by (auto simp: nsets-def)*  
**then obtain**  $a\ b\ c\ d$  **where**  $X = \{a,b,c,d\} \wedge a < b \wedge b < c \wedge c < d$   
**by** (*auto simp: ordered-nsets-4-eq*)  
**then show** *?thesis*  
**using**  $f2\ f\text{-less2}$  **by** (*auto simp: UU-def*)

**qed**

**then have**  $\exists N\ t. N \subseteq N1 \wedge \text{infinite } N \wedge t < \text{Suc}(\text{Suc } 0)$   
 $\wedge (\forall X. X \subseteq N \wedge \text{finite } X \wedge \text{card } X = 4 \implies f2\ X = t)$   
**using**  $\langle \text{infinite } N1 \rangle$  *Ramsey* [of *N1 4 f2 2*] **by** (*simp add: eval-nat-numeral*)  
**then obtain**  $N2\ j2$  **where**  $N2 \subseteq N1$  *infinite*  $N2$  **and**  $j2: j2 < \text{Suc}(\text{Suc } 0)$  **and**  
 $N2: \bigwedge A. A \in [N2]^4 \implies f2\ A = j2$

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

**define**  $f3$  **where**  $f3 \equiv (\lambda A::nat\ set.\ THE\ x.\ \exists a\ b\ c.\ A = \{a,b,c\} \wedge a < b \wedge b < c$   
 $\wedge x = f\ \{(a,b),(a,c)\})$

**have**  $f3: f3\ \{a,b,c\} = f\ \{(a,b),(a,c)\}$  **if**  $a < b\ b < c$  **for**  $a\ b\ c$   
**unfolding**  $f3\text{-def}$

**apply** (*rule theI2 [where a = f {(a,b), (a,c)}]*)

**using** *that by (force simp: insert-eq-iff split: if-split-asm)+*

**have**  $f3': f3\ \{a,b,c\} = f\ \{(a,b),(a,c)\}$  **if**  $a < c\ c < b$  **for**  $a\ b\ c$   
**using**  $f3$  [*of a c b*] *that*

**by** (*simp add: insert-commute*)

**have**  $f3\ X < Suc\ (Suc\ 0)$

**if** *finite X* **and**  $card\ X = 3$  **for**  $X$

**proof** –

**have**  $X \in [X]^3$

**using** *that by (auto simp: nsets-def)*

**then obtain**  $a\ b\ c$  **where**  $X = \{a,b,c\} \wedge a < b \wedge b < c$   
**by** (*auto simp: ordered-nsets-3-eq*)

**then show** *?thesis*

**using**  $f3\ f\text{-less2}$  **by** (*auto simp: UU-def*)

**qed**

**then have**  $\exists N\ t.\ N \subseteq N2 \wedge infinite\ N \wedge t < Suc\ (Suc\ 0)$   
 $\wedge (\forall X.\ X \subseteq N \wedge finite\ X \wedge card\ X = 3 \longrightarrow f3\ X = t)$

**using**  $\langle infinite\ N2 \rangle$  *Ramsey* [*of N2 3 f3 2*] **by** (*simp add: eval-nat-numeral*)

**then obtain**  $N3\ j3$  **where**  $N3 \subseteq N2$  *infinite N3* **and**  $j3: j3 < Suc\ (Suc\ 0)$  **and**  
 $N3: \bigwedge A.\ A \in [N3]^3 \implies f3\ A = j3$

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

**show** *thesis*

**proof**

**fix**  $k\ u$

**assume**  $k < 4$

**and**  $u:$  *form u k scheme u*  $\subseteq N3$

**and**  $UU:$   $u \in [UU]^2$

**then consider**  $(0)\ k=0 \mid (1)\ k=1 \mid (2)\ k=2 \mid (3)\ k=3$

**by** *linarith*

**then show**  $f\ u = [j0,j1,j2,j3] ! k$

**proof** *cases*

**case**  $0$

**have**  $N3 \subseteq N0$

**using**  $\langle N1 \subseteq N0 \rangle \langle N2 \subseteq N1 \rangle \langle N3 \subseteq N2 \rangle$  **by** *auto*

**then show** *?thesis*

**using**  $u\ 0$

**apply** (*auto simp: form-def form-split-def scheme-def simp flip: f0*)

**apply** (*force simp: nsets-def intro: N0*)

**done**

**next**

**case**  $1$

**have**  $N3 \subseteq N1$

```

    using ⟨N2 ⊆ N1⟩ ⟨N3 ⊆ N2⟩ by auto
  then show ?thesis
    using u 1
    apply (auto simp: form-def form-split-def scheme-def simp flip: f1)
    apply (force simp: nsets-def intro: N1)
  done
next
case 2
then show ?thesis
  using u ⟨N3 ⊆ N2⟩
  apply (auto simp: form-def form-split-def scheme-def nsets-def simp flip: f2)
  apply (force simp: nsets-def intro: N2)
  done
next
case 3
{ fix a b d
  assume {(a, b), (a, d)} ∈ [UU]2
  and *: a ∈ N3 b ∈ N3 d ∈ N3 b ≠ d
  then have a < b a < d
    by (auto simp: UU-def nsets-def)
  then have f {(a, b), (a, d)} = j3
    using *
    apply (auto simp: neq-iff simp flip: f3 f3')
    apply (force simp: nsets-def intro: N3)+
  done
}
then show ?thesis
  using u UU 3
  by (auto simp: form-def form-split-def scheme-def)
qed
qed (rule ⟨infinite N3⟩)
qed

```

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 –
  obtain f:: nat ⇒ nat where bij-betw f UNIV N strict-mono f
  using assms by (meson bij-enumerate enumerate-mono strict-monoI)
  then have iff[simp]: f x = f y ↔ x=y f x < f y ↔ x < y for x y
  by (simp-all add: strict-mono-eq strict-mono-less)
  have [simp]: f x ∈ N for x
  using bij-betw-apply [OF ⟨bij-betw f UNIV N⟩] by blast
  define M0 where M0 = (λi. (f(2*i), f(Suc(2*i)))) ‘{..m}
  define M1 where M1 = (λi. (f i, f(m+i))) ‘{..m}
  define M2 where M2 = (λi. (f i, f(2*m-i))) ‘{..m}
  define M3 where M3 = (λi. (f 0, f (Suc i))) ‘{..m}

```

```

consider (0)  $k=0$  | (1)  $k=1$  | (2)  $k=2$  | (3)  $k=3$ 
  using assms by linarith
then show thesis
proof cases
  case 0
  show ?thesis
  proof
    have inj-on ( $\lambda i. (f (2 * i), f (Suc (2 * i)))$ ) {.. $m$ }
      by (auto simp: inj-on-def)
    then show  $M0 \in [UU]^m$ 
      by (simp add: M0-def nsets-def card-image UU-def image-subset-iff)
  next
  fix  $u$ 
  assume  $u: (u: (nat \times nat) set) \in [M0]^2$ 
  then obtain  $x y$  where  $u = \{x, y\}$   $x \neq y$   $x \in M0$   $y \in M0$ 
    by (auto simp: nsets-2-eq)
  then obtain  $i j$  where  $i < j$   $j < m$  and ueq:  $u = \{(f(2*i), f(Suc(2*i))),$ 
( $f(2*j), f(Suc(2*j))\}$ 
    apply (clarsimp simp: M0-def)
    apply (metis (full-types) insert-commute less-linear)+
    done
  moreover have  $f (2 * i) \leq f (2 * j)$ 
    by (simp add: <i<j> less-imp-le-nat)
  ultimately show form  $u$   $k$ 
    apply (simp add: 0 form-def form-split-def nsets-def)
    apply (rule-tac x=f (2 * i) in exI)
    apply (rule-tac x=f (Suc (2 * i)) in exI)
    apply (rule-tac x=f (2 * j) in exI)
    apply (rule-tac x=f (Suc (2 * j)) in exI)
    apply auto
    done
  show scheme  $u \subseteq N$ 
    using ueq by (auto simp: scheme-def)
  qed
next
  case 1
  show ?thesis
  proof
    have inj-on ( $\lambda i. (f i, f(m+i))$ ) {.. $m$ }
      by (auto simp: inj-on-def)
    then show  $M1 \in [UU]^m$ 
      by (simp add: M1-def nsets-def card-image UU-def image-subset-iff)
  next
  fix  $u$ 
  assume  $u: (u: (nat \times nat) set) \in [M1]^2$ 
  then obtain  $x y$  where  $u = \{x, y\}$   $x \neq y$   $x \in M1$   $y \in M1$ 
    by (auto simp: nsets-2-eq)
  then obtain  $i j$  where  $i < j$   $j < m$  and ueq:  $u = \{(f i, f(m+i)), (f j, f(m+j))\}$ 
    apply (auto simp: M1-def)

```

```

    apply (metis (full-types) insert-commute less-linear)+
  done
then show form u k
  apply (simp add: 1 form-def form-split-def nsets-def)
  by (metis iff(2) nat-add-left-cancel-less nat-less-le trans-less-add1)
show scheme u  $\subseteq$  N
  using ueq by (auto simp: scheme-def)
qed
next
case 2
show ?thesis
proof
  have inj-on ( $\lambda i. (f\ i, f(2*m-i))$ )  $\{..<m\}$ 
    by (auto simp: inj-on-def)
  then show  $M2 \in [UU]^m$ 
    by (auto simp: M2-def nsets-def card-image UU-def image-subset-iff)
next
fix u
assume u: ( $u::(nat \times nat)\ set$ )  $\in [M2]^2$ 
then obtain x y where  $u = \{x,y\}$   $x \neq y$   $x \in M2$   $y \in M2$ 
  by (auto simp: nsets-2-eq)
then obtain i j where  $i < j < m$  and ueq:  $u = \{(f\ i, f(2*m-i)), (f\ j,$ 
 $f(2*m-j))\}$ 
  apply (auto simp: M2-def)
  apply (metis (full-types) insert-commute less-linear)+
  done
then show form u k
  apply (simp add: 2 form-def form-split-def nsets-def)
  apply (rule-tac  $x=f\ i$  in exI)
  apply (rule-tac  $x=f\ (2 * m - i)$  in exI)
  apply (rule-tac  $x=f\ j$  in exI)
  apply (rule-tac  $x=f\ (2 * m - j)$  in exI)
  apply (auto simp: less-imp-le-nat)
  done
show scheme u  $\subseteq$  N
  using ueq by (auto simp: scheme-def)
qed
next
case 3
show ?thesis
proof
  have inj-on ( $\lambda i. (f\ 0, f\ (Suc\ i))$ )  $\{..<m\}$ 
    by (auto simp: inj-on-def)
  then show  $M3 \in [UU]^m$ 
    by (auto simp: M3-def nsets-def card-image UU-def image-subset-iff)
next
fix u
assume u: ( $u::(nat \times nat)\ set$ )  $\in [M3]^2$ 
then obtain x y where  $u = \{x,y\}$   $x \neq y$   $x \in M3$   $y \in M3$ 

```



```

    by (auto simp: nsets-2-eq)
  then obtain  $i j$  where  $i < j < m$  and  $ueq: u = \{(f\ 0, f(Suc\ i)), (f\ 0, f(Suc\ j))\}$ 
    apply (auto simp: M3-def)
    apply (metis (full-types) insert-commute less-linear)+
    done
  then show  $form\ u\ k$ 
    by (fastforce simp: 3 form-def form-split-def nsets-def)
  show  $scheme\ u \subseteq N$ 
    using  $ueq$  by (auto simp: scheme-def)
qed
qed
qed

```

Lemma 2.5 of Jean A. Larson, *ibid.*

lemma *lemma-2-5*:

```

  assumes  $infinite\ N$ 
  obtains  $X$  where  $X \subseteq UU$   $ordertype\ X\ pair-less = \omega \uparrow 2$ 
     $\wedge u. u \in [X]^2 \implies (\exists k < 4. form\ u\ k) \wedge scheme\ u \subseteq N$ 
  proof -
    obtain  $C$ 
      where  $dis: pairwise\ (\lambda i\ j. disjnt\ (C\ i)\ (C\ j))\ UNIV$ 
        and  $N: (\bigcup i. C\ i) \subseteq N$  and  $infC: \bigwedge i::nat. infinite\ (C\ i)$ 
      using  $assms\ infinite-infinite-partition$  by blast
    then have  $\exists \varphi::nat \implies nat. inj\ \varphi \wedge range\ \varphi = C\ i \wedge strict-mono\ \varphi$  for  $i$ 
      by (metis  $nat-infinite-iff\ strict-mono-on-imp-inj-on$ )
    then obtain  $\varphi::[nat, nat] \implies nat$ 
      where  $\varphi: \bigwedge i. inj\ (\varphi\ i) \wedge range\ (\varphi\ i) = C\ i \wedge strict-mono\ (\varphi\ i)$ 
      by metis
    then have  $\pi-in-C\ [simp]: \varphi\ i\ j \in C\ i' \longleftrightarrow i'=i$  for  $i\ i'\ j$ 
      using  $dis$  by (fastforce simp: pairwise-def disjnt-def)
    have  $less-iff\ [simp]: \varphi\ i\ j' < \varphi\ i\ j \longleftrightarrow j' < j$  for  $i\ j'\ j$ 
      by (simp add:  $\varphi\ strict-mono-less$ )
    let  $?a = \varphi\ 0$ 
    define  $X$  where  $X \equiv \{(?a\ i, b) \mid i\ b. ?a\ i < b \wedge b \in C\ (Suc\ i)\}$ 
    show  $thesis$ 
  proof
    show  $X \subseteq UU$ 
      by (auto simp:  $X-def\ UU-def$ )
    show  $ordertype\ X\ pair-less = \omega \uparrow 2$ 
  proof (rule antisym)
    have  $ordertype\ X\ pair-less \leq ordertype\ UU\ pair-less$ 
      by (simp add:  $\langle X \subseteq UU \rangle\ ordertype-mono$ )
    then show  $ordertype\ X\ pair-less \leq \omega \uparrow 2$ 
      using  $ordertype-UU-\omega 2$  by auto
    define  $\pi$  where  $\pi \equiv \lambda(i, j::nat). (?a\ i, \varphi\ (Suc\ i)\ (?a\ j))$ 
    have  $\bigwedge i\ j. i < j \implies \varphi\ 0\ i < \varphi\ (Suc\ i)\ (\varphi\ 0\ j)$ 
      by (meson  $\varphi\ le-less-trans\ less-iff\ strict-mono-imp-increasing$ )
    then have  $subX: \pi\ ' UU \subseteq X$ 

```

```

    by (auto simp: UU-def  $\pi$ -def X-def)
  then have ordertype ( $\pi$  ' UU) pair-less  $\leq$  ordertype X pair-less
    by (simp add: ordertype-mono)
  moreover have ordertype ( $\pi$  ' UU) pair-less = ordertype UU pair-less
  proof (rule ordertype-inc-eq)
    show ( $\pi$  x,  $\pi$  y)  $\in$  pair-less
      if  $x \in$  UU  $y \in$  UU and  $(x, y) \in$  pair-less for  $x$   $y$ 
      using that by (auto simp: UU-def  $\pi$ -def pair-less-def)
    qed auto
  ultimately show  $\omega \uparrow 2 \leq$  ordertype X pair-less
    using ordertype-UU- $\omega 2$  by simp
  qed
next
fix U
assume  $U \in [X]^2$ 
then obtain a b c d where Ueq:  $U = \{(a,b),(c,d)\}$  and ne:  $(a,b) \neq (c,d)$  and
inX:  $(a,b) \in X$   $(c,d) \in X$  and  $a \leq c$ 
apply (auto simp: nsets-def subset-iff eval-nat-numeral card-Suc-eq Set.doubleton-eq-iff)
  apply (metis nat-le-linear)+
  done
show  $(\exists k < 4. \text{form } U k) \wedge \text{scheme } U \subseteq N$ 
proof
  show scheme  $U \subseteq N$ 
    using inX N  $\varphi$  by (fastforce simp: scheme-def Ueq X-def)
next
consider  $a < c \mid a = c \wedge b \neq d$ 
  using  $\langle a \leq c \rangle$  ne nat-less-le by blast
then show  $\exists k < 4. \text{form } U k$ 
proof cases
  case 1
  have *:  $a < b \mid b \neq c \mid c < d$ 
    using inX by (auto simp: X-def)
  moreover have  $\llbracket a < c; c < b; \neg d < b \rrbracket \implies b < d$ 
    using inX apply (clarsimp simp: X-def not-less)
    by (metis  $\varphi$   $\pi$ -in-C imageE nat.inject nat-less-le)
  ultimately consider (k0)  $a < b \wedge b < c \wedge c < d \mid$  (k1)  $a < c \wedge c < b \wedge b < d \mid$ 
(k2)  $a < c \wedge c < d \wedge d < b$ 
    using 1 less-linear by blast
  then show ?thesis
proof cases
  case k0
  then have form U 0
    unfolding form-def form-split-def using Ueq  $\langle a \leq c \rangle$  by blast
  then show ?thesis by force
next
  case k1
  then have form U 1
    unfolding form-def form-split-def using Ueq  $\langle a \leq c \rangle$  by blast
  then show ?thesis by force

```

```

next
  case k2
  then have form U 2
    unfolding form-def form-split-def using Ueq ⟨a ≤ c⟩ by blast
  then show ?thesis by force
qed
next
  case 2
  then have form-split a b c d 3
    by (auto simp: form-split-def)
  then show ?thesis
    using Ueq form-def leI by force
qed
qed
qed
qed

```

Theorem 2.1 of Jean A. Larson, *ibid.*

**lemma** *Specker-aux*:

```

assumes α ∈ elts ω
shows partn-1st pair-less UU [ω↑2, α] 2
unfolding partn-1st-def
proof clarsimp
  fix f
  assume f: f ∈ [UU]2 → {..<Suc (Suc 0)}
  let ?P0 = ∃ X ⊆ UU. ordertype X pair-less = ω↑2 ∧ f ‘ [X]2 ⊆ {0}
  let ?P1 = ∃ M ⊆ UU. ordertype M pair-less = α ∧ f ‘ [M]2 ⊆ {1}
  have †: ?P0 ∨ ?P1
  proof (rule disjCI)
    assume ¬ ?P1
    then have not1: ∧M. [M ⊆ UU; ordertype M pair-less = α] ⇒ ∃ x ∈ [M]2. f
      x ≠ Suc 0
    by auto
    obtain m where m: α = ord-of-nat m
    using assms elts-ω by auto
    then have f-eq-0: M ∈ [UU]m ⇒ ∃ x ∈ [M]2. f x = 0 for M
    using not1 [of M] finite-ordertype-eq-card [of M pair-less m] f
    apply (clarsimp simp: nsets-def eval-nat-numeral Pi-def)
    by (meson less-Suc0 not-less-less-Suc-eq subset-trans)
    obtain N js where infinite N and N: ∧k u. [k < 4; u ∈ [UU]2; form u k;
      scheme u ⊆ N] ⇒ f u = js!k
    using f lemma-2-3 by blast
    obtain M0 where M0: M0 ∈ [UU]m ∧ u. u ∈ [M0]2 ⇒ form u 0 ∧ u. u ∈
      [M0]2 ⇒ scheme u ⊆ N
    by (rule lemma-2-4 [OF ⟨infinite N⟩]) auto
    obtain M1 where M1: M1 ∈ [UU]m ∧ u. u ∈ [M1]2 ⇒ form u 1 ∧ u. u ∈
      [M1]2 ⇒ scheme u ⊆ N
    by (rule lemma-2-4 [OF ⟨infinite N⟩]) auto
    obtain M2 where M2: M2 ∈ [UU]m ∧ u. u ∈ [M2]2 ⇒ form u 2 ∧ u. u ∈

```

```

[M2]2 ⇒ scheme u ⊆ N
  by (rule lemma-2-4 [OF <infinite N>]) auto
  obtain M3 where M3: M3 ∈ [UU]m ∧ u. u ∈ [M3]2 ⇒ form u 3 ∧ u. u ∈
[M3]2 ⇒ scheme u ⊆ N
  by (rule lemma-2-4 [OF <infinite N>]) auto
  have js!0 = 0
  using N [of 0] M0 f-eq-0 [of M0] by (force simp: nsets-def eval-nat-numeral)
  moreover have js!1 = 0
  using N [of 1] M1 f-eq-0 [of M1] by (force simp: nsets-def eval-nat-numeral)
  moreover have js!2 = 0
  using N [of 2] M2 f-eq-0 [of M2] by (force simp: nsets-def eval-nat-numeral)
  moreover have js!3 = 0
  using N [of 3] M3 f-eq-0 [of M3] by (force simp: nsets-def eval-nat-numeral)
  ultimately have js0: js!k = 0 if k < 4 for k
  using that by (auto simp: eval-nat-numeral less-Suc-eq)
  obtain X where X ⊆ UU and otX: ordertype X pair-less = ω↑2
  and X: ∧u. u ∈ [X]2 ⇒ (∃ k < 4. form u k) ∧ scheme u ⊆ N
  using <infinite N> lemma-2-5 by auto
  moreover have f ' [X]2 ⊆ {0}
  proof (clarsimp simp: image-subset-iff)
    fix u
    assume u: u ∈ [X]2
    then have u-UU2: u ∈ [UU]2
      using <X ⊆ UU> nsets-mono by blast
    show f u = 0
      using X u N [OF - u-UU2] js0 by auto
  qed
  ultimately show ∃ X ⊆ UU. ordertype X pair-less = ω↑2 ∧ f ' [X]2 ⊆ {0}
  by blast
qed
then show ∃ i < Suc (Suc 0). ∃ H ⊆ UU. ordertype H pair-less = [ω↑2, α] ! i ∧ f
' [H]2 ⊆ {i}
  by (metis One-nat-def lessI nth-Cons-0 nth-Cons-Suc zero-less-Suc)
qed

theorem Specker: α ∈ elts ω ⇒ partn-lst-VWF (ω↑2) [ω↑2,α] 2
  using partn-lst-imp-partn-lst-VWF-eq [OF Specker-aux] ordertype-UU-ω2 wf-pair-less
  by blast

end
theory Erdos-Milner
  imports Partitions

begin

```

## 2.5 Erdos-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**  $D: D \subseteq \text{elts } \beta$  **and**  $\text{Ord } \beta$

**obtains**  $L$  **where**  $\bigcup (\text{List.set } L) = D \wedge X. X \in \text{List.set } L \implies \text{indecomposable}$   
( $tp\ X$ )

**and**  $\bigwedge M. \llbracket M \subseteq D; \bigwedge X. X \in \text{List.set } L \implies tp\ (M \cap X) \geq tp\ X \rrbracket \implies tp\ M = tp\ D$

**proof** –

**define**  $\varphi$  **where**  $\varphi \equiv \text{inv-into } D\ (\text{ordermap } D\ \text{VWF})$

**then have**  $\text{bij-}\varphi: \text{bij-betw } \varphi\ (\text{elts } (tp\ D))\ D$

**using**  $D$  *bij-betw-inv-into down ordermap-bij* **by** *blast*

**have**  $\varphi\text{-cancel-left}: \bigwedge d. d \in D \implies \varphi\ (\text{ordermap } D\ \text{VWF } d) = d$

**by** (*metis*  $D$   $\varphi\text{-def}$  *bij-betw-inv-into-left down ordermap-bij total-on-VWF wf-VWF*)

**have**  $\varphi\text{-cancel-right}: \bigwedge \gamma. \gamma \in \text{elts } (tp\ D) \implies \text{ordermap } D\ \text{VWF } (\varphi\ \gamma) = \gamma$

**by** (*metis*  $\varphi\text{-def}$  *f-inv-into-f ordermap-surj subsetD*)

**have** *small*  $D\ D \subseteq ON$

**using** *assms down elts-subset-ON [of  $\beta$ ]* **by** *auto*

**then have**  $\varphi\text{-less-iff}: \bigwedge \gamma\ \delta. \llbracket \gamma \in \text{elts } (tp\ D); \delta \in \text{elts } (tp\ D) \rrbracket \implies \varphi\ \gamma < \varphi\ \delta$   
 $\longleftrightarrow \gamma < \delta$

**by** (*metis*  $\varphi\text{-def}$  *inv-ordermap-VWF-mono-iff inv-ordermap-mono-eq less-V-def*)

**obtain**  $\alpha s$  **where**  $\text{List.set } \alpha s \subseteq ON$  **and**  $\omega\text{-dec } \alpha s$  **and**  $tpD\text{-eq}: tp\ D = \omega\text{-sum } \alpha s$

**using** *Ord-ordertype  $\omega\text{-nf-exists}$*  **by** *blast* — Cantor normal form of the ordertype

**have** *ord [simp]:*  $\text{Ord } (\alpha s ! k)\ \text{Ord } (\omega\text{-sum } (\text{take } k\ \alpha s))$  **if**  $k < \text{length } \alpha s$  **for**  $k$

**using** *that*  $\langle \text{list.set } \alpha s \subseteq ON \rangle$

**by** (*auto simp: dual-order.trans set-take-subset elim: ON-imp-Ord*)

**define**  $E$  **where**  $E \equiv \lambda k. \text{lift } (\omega\text{-sum } (\text{take } k\ \alpha s))\ (\omega \uparrow (\alpha s ! k))$

**define**  $L$  **where**  $L \equiv \text{map } (\lambda k. \varphi\ ' (\text{elts } (E\ k)))\ [0..<\text{length } \alpha s]$

**have** *lengthL [simp]:*  $\text{length } L = \text{length } \alpha s$

**by** (*simp add: L-def*)

**have** *in-elts-E-less:*  $\text{elts } (E\ k') \ll \text{elts } (E\ k)$  **if**  $k' < k < \text{length } \alpha s$  **for**  $k\ k'$

— The ordinals have been partitioned into disjoint intervals

**proof** –

**have** *ord $\omega$ :*  $\text{Ord } (\omega \uparrow \alpha s ! k')$

**using** *that* **by** *auto*

**from** *that id-take-nth-drop [of  $k'$  take  $k$   $\alpha s$ ]*

**obtain**  $l$  **where**  $\text{take } k\ \alpha s = \text{take } k'\ \alpha s @ (\alpha s ! k') \# l$

**by** (*simp add: min-def*)

**then show** *?thesis*

**using** *that unfolding E-def lift-def less-sets-def*

**by** (*auto dest!: OrdmemD [OF ord $\omega$ ] intro: less-le-trans*)

**qed**

**have** *elts-E:*  $\text{elts } (E\ k) \subseteq \text{elts } (\omega\text{-sum } \alpha s)$

**if**  $k < \text{length } \alpha s$  **for**  $k$

**proof** –

**have**  $\omega \uparrow (\alpha s!k) \leq \omega\text{-sum } (\text{drop } k \ \alpha s)$   
**by** (*metis that Cons-nth-drop-Suc  $\omega\text{-sum-Cons add-le-cancel-left0}$* )  
**then have**  $(+) (\omega\text{-sum } (\text{take } k \ \alpha s)) \text{ 'elts } (\omega \uparrow (\alpha s!k)) \subseteq \text{elts } (\omega\text{-sum } (\text{take } k \ \alpha s) + \omega\text{-sum } (\text{drop } k \ \alpha s))$   
**by** *blast*  
**also have**  $\dots = \text{elts } (\omega\text{-sum } \alpha s)$   
**using**  $\omega\text{-sum-take-drop}$  **by** *auto*  
**finally show** *?thesis*  
**by** (*simp add: lift-def E-def*)  
**qed**  
**have**  $\omega\text{-sum-in-tpD: } \omega\text{-sum } (\text{take } k \ \alpha s) + \gamma \in \text{elts } (tp \ D)$   
**if**  $\gamma \in \text{elts } (\omega \uparrow \alpha s!k) \ k < \text{length } \alpha s$  **for**  $\gamma \ k$   
**using**  $\text{elts-E lift-def that tpD-eq}$  **by** (*auto simp: E-def*)  
**have**  $Ord\text{-}\varphi: Ord \ (\varphi \ (\omega\text{-sum } (\text{take } k \ \alpha s) + \gamma))$   
**if**  $\gamma \in \text{elts } (\omega \uparrow \alpha s!k) \ k < \text{length } \alpha s$  **for**  $\gamma \ k$   
**using**  $\omega\text{-sum-in-tpD} \langle D \subseteq ON \rangle \text{ bij-}\varphi \text{ bij-betw-imp-surj-on that}$  **by** *fastforce*  
**define**  $\pi$  **where**  $\pi \equiv \lambda k. ((\lambda y. \text{odiff } y \ (\omega\text{-sum } (\text{take } k \ \alpha s))) \circ \text{ordermap } D \ VWF)$   
— mapping the segments of  $D$  into some power of  $\omega$   
**have**  $\text{bij-}\pi: \text{bij-betw } (\pi \ k) \ (\varphi \ \text{ 'elts } (E \ k)) \ (\text{elts } (\omega \uparrow (\alpha s!k)))$   
**if**  $k < \text{length } \alpha s$  **for**  $k$   
**using that by** (*auto simp: bij-betw-def  $\pi\text{-def E-def inj-on-def lift-def image-comp}$*   
 $\omega\text{-sum-in-tpD } \varphi\text{-cancel-right that}$ )  
**have**  $\pi\text{-iff: } \pi \ k \ x < \pi \ k \ y \longleftrightarrow (x, y) \in VWF$   
**if**  $x \in \varphi \ \text{ 'elts } (E \ k) \ y \in \varphi \ \text{ 'elts } (E \ k) \ k < \text{length } \alpha s$  **for**  $k \ x \ y$   
**using that**  
**by** (*auto simp:  $\pi\text{-def E-def lift-def } \omega\text{-sum-in-tpD } \varphi\text{-cancel-right Ord-}\varphi \ \varphi\text{-less-iff}$* )  
**have**  $tp\text{-E-eq [simp]: } tp \ (\varphi \ \text{ 'elts } (E \ k)) = \omega \uparrow (\alpha s!k)$   
**if**  $k: k < \text{length } \alpha s$  **for**  $k$   
**by** (*metis Ord- $\omega$  Ord-oexp  $\pi\text{-iff bij-}\pi \text{ ord}(1) \text{ ordertype-VWF-eq-iff replacement}$*   
*small-elts that*)  
**have**  $tp\text{-L-eq [simp]: } tp \ (L!k) = \omega \uparrow (\alpha s!k)$  **if**  $k < \text{length } \alpha s$  **for**  $k$   
**by** (*simp add: L-def that*)  
**have**  $UL\text{-eq-D: } \bigcup (list.set \ L) = D$   
**proof** (*rule antisym*)  
**show**  $\bigcup (list.set \ L) \subseteq D$   
**by** (*force simp: L-def tpD-eq bij-betw-apply [OF  $\text{bij-}\varphi$ ] dest: elts-E*)  
**show**  $D \subseteq \bigcup (list.set \ L)$   
**proof**  
**fix**  $\delta$   
**assume**  $\delta \in D$   
**then have**  $\text{ordermap } D \ VWF \ \delta \in \text{elts } (\omega\text{-sum } \alpha s)$   
**by** (*metis  $\langle \text{small } D \rangle \text{ ordermap-in-ordertype tpD-eq}$* )  
**then show**  $\delta \in \bigcup (list.set \ L)$   
**using**  $\langle \delta \in D \rangle \varphi\text{-cancel-left in-elts-}\omega\text{-sum}$   
**by** (*fastforce simp: L-def E-def image-iff lift-def*)  
**qed**  
**qed**  
**show** *thesis*  
**proof**

```

show indecomposable (tp X) if X ∈ list.set L for X
  using that by (auto simp: L-def indecomposable-ω-power)
next
fix M
assume M ⊆ D and *:  $\bigwedge X. X \in \text{list.set } L \implies \text{tp } X \leq \text{tp } (M \cap X)$ 
show tp M = tp D
proof (rule antisym)
  show tp M ≤ tp D
    by (simp add: ⟨M ⊆ D⟩ ⟨small D⟩ ordertype-VWF-mono)
  define σ where σ ≡ λX. inv-into (M ∩ X) (ordermap (M ∩ X) VWF)
    — The bijection from an ω-power into the appropriate
segment of M
  have bij-σ: bij-betw (σ X) (elts (tp (M ∩ X))) (M ∩ X) for X
    unfolding σ-def
      by (meson ⟨M ⊆ D⟩ ⟨small D⟩ bij-betw-inv-into inf-le1 ordermap-bij
smaller-than-small total-on-VWF wf-VWF)
  have Ord-σ: Ord (σ X α) if α ∈ elts (tp (M ∩ X)) for α X
    using ⟨D ⊆ ON⟩ ⟨M ⊆ D⟩ bij-betw-apply [OF bij-σ] that by blast
  have σ-cancel-right:  $\bigwedge \gamma X. \gamma \in \text{elts } (\text{tp } (M \cap X)) \implies \text{ordermap } (M \cap X)$ 
VWF (σ X γ) = γ
    by (metis σ-def f-inv-into-f ordermap-surj subsetD)
  have smM: small (M ∩ X) for X
    by (meson ⟨M ⊆ D⟩ ⟨small D⟩ inf-le1 subset-iff-less-eq-V)
  have  $\exists k < \text{length } \alpha s. \gamma \in \text{elts } (E k)$  if γ: γ ∈ elts (tp D) for γ
proof —
  obtain X where X ∈ set L and X: φ γ ∈ X
    by (metis UL-eq-D γ Union-iff φ-def in-mono inv-into-into ordermap-surj)
  then have  $\exists k < \text{length } \alpha s. \gamma \in \text{elts } (E k) \wedge X = \varphi \text{ ` } \text{elts } (E k)$ 
    apply (clarsimp simp: L-def)
    by (metis γ φ-cancel-right elts-E in-mono tpD-eq)
  then show ?thesis
    by blast
qed
then obtain K where K:  $\bigwedge \gamma. \gamma \in \text{elts } (\text{tp } D) \implies K \gamma < \text{length } \alpha s \wedge \gamma \in$ 
elts (E (K γ))
    by metis — The index from tp D to the appropriate segment number
  define ψ where ψ ≡ λd. σ (L ! K (ordermap D VWF d)) (π (K (ordermap
D VWF d)) d)
show tp D ≤ tp M
proof (rule ordertype-inc-le)
  show small D small M
    using ⟨M ⊆ D⟩ ⟨small D⟩ subset-iff-less-eq-V by auto
next
fix d' d
assume xy: d' ∈ D d ∈ D and (d', d) ∈ VWF
then have d' < d
    using ON-imp-Ord ⟨D ⊆ ON⟩ by auto
let ?γ' = ordermap D VWF d'
let ?γ = ordermap D VWF d

```

**have**  $len'$ :  $K \ ?\gamma' < length \ \alpha s$  **and**  $elts'$ :  $?\gamma' \in elts \ (E \ (K \ ?\gamma'))$   
**and**  $len$ :  $K \ ?\gamma < length \ \alpha s$  **and**  $elts$ :  $?\gamma \in elts \ (E \ (K \ ?\gamma))$   
**using**  $K \ \langle d' \in D \rangle \ \langle d \in D \rangle$  **by** (*auto simp*:  $\langle small \ D \rangle$  *ordermap-in-ordertype*)  
**have**  $Ord\text{-}\sigma L$ :  $Ord \ (\sigma \ (L!k) \ (\pi \ k \ d))$  **if**  $d \in \varphi \ \langle elts \ (E \ k) \ k < length \ \alpha s$  **for**  
 $k \ d$   
**by** (*metis* (*mono-tags*) \*  $Ord\text{-}\sigma$  *bij- $\pi$*  *bij-betw-apply lengthL nth-mem* *that*  
*tp-L-eq vsubsetD*)  
**have**  $?\gamma' < ?\gamma$   
**by** (*simp add*:  $\langle (d', d) \in VWF \rangle \ \langle small \ D \rangle$  *ordermap-mono-less xy*)  
**then have**  $K \ ?\gamma' \leq K \ ?\gamma$   
**using**  $elts' \ elts \ in\text{-}elts\text{-}E\text{-}less$  **by** (*meson leI len' less-asym less-sets-def*)  
**then consider** (*less*)  $K \ ?\gamma' < K \ ?\gamma \mid$  (*equal*)  $K \ ?\gamma' = K \ ?\gamma$   
**by** *linarith*  
**then have**  $\sigma \ (L!K \ ?\gamma') \ (\pi \ (K \ ?\gamma') \ d') < \sigma \ (L!K \ ?\gamma) \ (\pi \ (K \ ?\gamma) \ d)$   
**proof cases**  
**case less**  
**obtain**  $\dagger$ :  $\sigma \ (L!K \ ?\gamma') \ (\pi \ (K \ ?\gamma') \ d') \in M \cap L!K \ ?\gamma' \ \sigma \ (L!K \ ?\gamma) \ (\pi$   
 $(K \ ?\gamma) \ d) \in M \cap L!K \ ?\gamma$   
**using**  $elts' \ elts \ len' \ len$  \* [*THEN vsubsetD*]  
**by** (*metis lengthL  $\varphi$ -cancel-left bij- $\pi$  bij- $\sigma$  bij-betw-imp-surj-on imageI*  
*nth-mem tp-L-eq xy*)  
**then have** *ordermap D VWF*  $(\sigma \ (L!K \ ?\gamma') \ (\pi \ (K \ ?\gamma') \ d')) \in elts \ (E \ (K$   
 $?\gamma'))$  *ordermap D VWF*  $(\sigma \ (L!K \ ?\gamma) \ (\pi \ (K \ ?\gamma) \ d)) \in elts \ (E \ (K \ ?\gamma))$   
**using**  $len' \ len \ elts\text{-}E \ tpD\text{-}eq$   
**by** (*fastforce simp*:  $L\text{-}def \ \varphi\text{-}cancel\text{-}right$ )  
**then have** *ordermap D VWF*  $(\sigma \ (L!K \ ?\gamma') \ (\pi \ (K \ ?\gamma') \ d')) < \text{ordermap}$   
 $D \ VWF \ (\sigma \ (L!K \ ?\gamma) \ (\pi \ (K \ ?\gamma) \ d))$   
**using** *in-elts-E-less len less* **by** (*meson less-sets-def*)  
**moreover have**  $\sigma \ (L!K \ ?\gamma') \ (\pi \ (K \ ?\gamma') \ d') \in D \ \sigma \ (L!K \ ?\gamma) \ (\pi \ (K \ ?\gamma)$   
 $d) \in D$   
**using**  $\langle M \subseteq D \rangle \ \dagger$  **by** *auto*  
**ultimately show** *?thesis*  
**by** (*metis*  $\langle small \ D \rangle \ \varphi\text{-}cancel\text{-}left \ \varphi\text{-}less\text{-}iff \ \text{ordermap-in-ordertype}$ )  
**next**  
**case equal**  
**have**  $\sigma\text{-}less$ :  $\bigwedge X \ \gamma \ \delta. \llbracket \gamma < \delta; \gamma \in elts \ (tp \ (M \cap X)); \delta \in elts \ (tp \ (M \cap$   
 $X)) \rrbracket$   
 $\implies \sigma \ X \ \gamma < \sigma \ X \ \delta$   
**by** (*metis*  $\langle D \subseteq ON \rangle \ \langle M \subseteq D \rangle \ \sigma\text{-}def \ \text{dual-order.trans inv-ordermap-VWF-strict-mono-iff}$   
*le-infI1 smM*)  
**have**  $\pi \ (K \ ?\gamma) \ d' < \pi \ (K \ ?\gamma) \ d$   
**by** (*metis equal*  $\langle (d', d) \in VWF \rangle \ \varphi\text{-}cancel\text{-}left \ \pi\text{-iff} \ elts \ elts' \ imageI \ len$   
 $xy$ )  
**then show** *?thesis*  
**unfolding equal**  
**by** (*metis* \* [*THEN vsubsetD*] *lengthL  $\varphi$ -cancel-left  $\sigma$ -less bij- $\pi$*   
*bij-betw-imp-surj-on elts elts' image-eqI len local.equal nth-mem tp-L-eq*  
 $xy$ )  
**qed**



**moreover have**  $Ord (\sigma (L ! K ?\gamma') (\pi (K ?\gamma') d')) Ord (\sigma (L ! K ?\gamma) (\pi (K ?\gamma) d))$   
**using** *Ord- $\sigma$ L  $\varphi$ -cancel-left elts len elts' len' xy by fastforce+*  
**ultimately show**  $(\psi d', \psi d) \in VWF$   
**by** (*simp add:  $\psi$ -def*)  
**next**  
**show**  $\psi \langle D \subseteq M$   
**proof** (*clarsimp simp:  $\psi$ -def*)  
**fix**  $d$   
**assume**  $d \in D$   
**let**  $?\gamma = ordermap D VWF d$   
**have**  $len: K ?\gamma < length \alpha$  **and**  $elts: ?\gamma \in elts (E (K ?\gamma))$   
**using**  $K \langle d \in D \rangle$  **by** (*auto simp:  $\langle$ small  $D \rangle$  ordermap-in-ordertype*)  
**have**  $\pi (K ?\gamma) d \in elts (tp (L! (K ?\gamma)))$   
**using**  $bij-\pi [OF len] \langle d \in D \rangle$   
**by** (*metis  $\varphi$ -cancel-left bij-betw-apply elts imageI len tp-L-eq*)  
**then show**  $\sigma (L ! K (ordermap D VWF d)) (\pi (K (ordermap D VWF d))) d) \in M$   
**by** (*metis \* lengthL Int-iff bij- $\sigma$  bij-betw-apply len nth-mem vsubsetD*)  
**qed**  
**qed** *auto*  
**qed**  
**qed** (*simp add: UL-eq-D*)  
**qed**

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

**proposition** *indecomposable-imp-Ex-less-sets:*

**assumes** *indec: indecomposable  $\alpha$  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** –

**have**  $Ord \alpha$

**using** *indec indecomposable-imp-Ord by blast*

**have**  $Limit \alpha$

**by** (*meson  $\omega$ -gt1 assms indec indecomposable-imp-Limit less-le-trans*)

**define**  $\varphi$  **where**  $\varphi \equiv inv-into A (ordermap A VWF)$

**then have**  $bij-\varphi: bij-betw \varphi (elts \alpha) A$

**using**  $A$  *bij-betw-inv-into down ordermap-bij by blast*

**have**  $bij-om: bij-betw (ordermap A VWF) A (elts \alpha)$

**using**  $A$  *down ordermap-bij by blast*

**define**  $\gamma$  **where**  $\gamma \equiv ordermap A VWF x$

**have**  $\gamma: \gamma \in elts \alpha$

**unfolding**  $\gamma$ -def **using**  $A \langle x \in A \rangle$  *down by auto*

**then have**  $Ord \gamma$

**using** *Ord-in-Ord  $\langle Ord \alpha \rangle$  by blast*

**define**  $B$  **where**  $B \equiv \varphi \langle elts (succ \gamma) \rangle$

**have**  $B: \{y \in A. ordermap A VWF y \leq ordermap A VWF x\} \subseteq B$

```

apply (clarsimp simp add: B-def  $\gamma$ -def image-iff  $\varphi$ -def)
by (metis Ord-linear Ord-ordermap OrdmemD bij-betw-inv-into-left bij-om leD)
show thesis
proof
  have small A1
    by (meson  $\langle$ small A $\rangle$  A1 smaller-than-small)
  then have tp (A1 - B)  $\leq$  tp A1
    by (simp add: ordertype-VWF-mono)
  moreover have tp (A1 - B)  $\geq$   $\alpha$ 
  proof -
    have  $\neg$  ( $\alpha \leq$  tp B)
      unfolding B-def
    proof (subst ordertype-VWF-inc-eq)
      show elts (succ  $\gamma$ )  $\subseteq$  ON
        by (auto simp:  $\gamma$ -def ordertype-VWF-inc-eq intro: Ord-in-Ord)
      have sub: elts (succ  $\gamma$ )  $\subseteq$  elts  $\alpha$ 
        using Ord-trans  $\gamma$   $\langle$ Ord  $\alpha$  $\rangle$  by auto
      then show  $\varphi$  ' elts (succ  $\gamma$ )  $\subseteq$  ON
        using  $\langle$ A  $\subseteq$  ON $\rangle$  bij- $\varphi$  bij-betw-imp-surj-on by blast
      have succ  $\gamma \in$  elts  $\alpha$ 
        using  $\gamma$  Limit-def  $\langle$ Limit  $\alpha$  $\rangle$  by blast
      with A sub show  $\varphi$  u <  $\varphi$  v
        if u  $\in$  elts (succ  $\gamma$ ) and v  $\in$  elts (succ  $\gamma$ ) and u < v for u v
        by (metis  $\varphi$ -def inv-ordermap-VWF-strict-mono-iff subsetD that)
      show  $\neg$   $\alpha \leq$  tp (elts (succ  $\gamma$ ))
        by (metis Limit-def Ord-succ  $\gamma$   $\langle$ Limit  $\alpha$  $\rangle$   $\langle$ Ord  $\gamma$  $\rangle$  mem-not-refl order-
type-eq-Ord vsubsetD)
    qed auto
  then show ?thesis
    using indecomposable-ordertype-ge [OF indec, of A1 B]  $\langle$ small A1 $\rangle$  A1 by
(auto simp: B-def)
  qed
  ultimately show tp (A1 - B) =  $\alpha$ 
    using A1 by blast
  have {x}  $\ll$  (A - B)
    using assms B
  apply (clarsimp simp: less-sets-def  $\varphi$ -def subset-iff)
  by (metis Ord-not-le VWF-iff-Ord-less less-V-def order-refl ordermap-mono-less
trans-VWF wf-VWF)
  with  $\langle$ A1  $\subseteq$  A $\rangle$  show {x}  $\ll$  (A1 - B) by auto
  qed auto
qed

```

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  $\alpha$  and k > 1 Ord  $\gamma$  and  $\beta$ :  $\beta \in$  elts  $\omega$ 1
shows partn-lst-VWF ( $\alpha * \beta$ ) [ord-of-nat (2*k), min  $\gamma$  ( $\omega * \beta$ )] 2
proof (cases  $\alpha \leq 1 \vee \beta = 0$ )

```

```

case True
have Ord  $\alpha$ 
  using indec indecomposable-def by blast
show ?thesis
proof (cases  $\beta=0$ )
  case True then show ?thesis
    by (simp add: partn-lst-triv0 [where  $i=1$ ])
  next
  case False
  then consider (0)  $\alpha=0$  | (1)  $\alpha=1$ 
  by (metis Ord-0 OrdmemD True  $\langle$ Ord  $\alpha$  $\rangle$  mem-0-Ord one-V-def order.antisym
succ-le-iff)
  then show ?thesis
  proof cases
    case 0
    with part show ?thesis
      by (force simp add: partn-lst-def nsets-empty-iff less-Suc-eq)
    next
    case 1
    then obtain Ord  $\beta$ 
      by (meson ON-imp-Ord Ord- $\omega$ 1 True  $\beta$  elts-subset-ON)
    then obtain  $i$  where  $i < \text{Suc } (\text{Suc } 0)$  [ord-of-nat  $k, \gamma$ ] !  $i \leq \alpha$ 
      using partn-lst-VWF-nontriv [OF part] 1 by auto
    then have  $\gamma \leq 1$ 
      using  $\langle \alpha=1 \rangle \langle k > 1 \rangle$  by (fastforce simp: less-Suc-eq)
    then have  $\min \gamma (\omega * \beta) \leq 1$ 
      by (metis Ord-1 Ord- $\omega$  Ord-linear-le Ord-mult  $\langle$ Ord  $\beta$  $\rangle$  min-def order-trans)
    then show ?thesis
      using False by (auto simp: True  $\langle$ Ord  $\beta$  $\rangle \langle \beta \neq 0 \rangle \langle \alpha=1 \rangle$  intro!: partn-lst-triv1
[where  $i=1$ ])
  qed
qed
next
case False
then have  $\alpha \geq \omega$ 
  by (meson Ord-1 Ord-not-less indec indecomposable-def indecomposable-imp-Limit
omega-le-Limit)
have  $0 \in \text{elts } \beta$   $\beta \neq 0$ 
  using False Ord- $\omega$ 1 Ord-in-Ord  $\beta$  mem-0-Ord by blast+
show ?thesis
  unfolding partn-lst-def
proof clarsimp
  fix  $f$ 
  assume  $f \in [\text{elts } (\alpha * \beta)]^2 \rightarrow \{.. < \text{Suc } (\text{Suc } 0)\}$ 
  then have  $f: f \in [\text{elts } (\alpha * \beta)]^2 \rightarrow \{.. < 2::\text{nat}\}$ 
    by (simp add: eval-nat-numeral)
  obtain ord [iff]: Ord  $\alpha$  Ord  $\beta$  Ord  $(\alpha * \beta)$ 
    using Ord- $\omega$ 1 Ord-in-Ord  $\beta$  indec indecomposable-imp-Ord Ord-mult by blast
  have *: False

```

**if**  $i$  [rule-format]:  $\forall H. tp\ H = ord\text{-of-nat}\ (2*k) \longrightarrow H \subseteq elts\ (\alpha*\beta) \longrightarrow \neg f$   
 $\text{' } [H]^2 \subseteq \{0\}$   
**and**  $ii$  [rule-format]:  $\forall H. tp\ H = \gamma \longrightarrow H \subseteq elts\ (\alpha*\beta) \longrightarrow \neg f \text{' } [H]^2 \subseteq$   
 $\{1\}$   
**and**  $iii$  [rule-format]:  $\forall H. tp\ H = (\omega*\beta) \longrightarrow H \subseteq elts\ (\alpha*\beta) \longrightarrow \neg f \text{' } [H]^2$   
 $\subseteq \{1\}$   
**proof** –  
**have**  $Ak0$ :  $\exists X \in [A]^k. f \text{' } [X]^2 \subseteq \{0\}$  — remark (8) about  $A \subseteq S$   
**if**  $A-\alpha\beta$ :  $A \subseteq elts\ (\alpha*\beta)$  **and**  $ot$ :  $tp\ A \geq \alpha$  **for**  $A$   
**proof** –  
**let**  $?g = inv\text{-into}\ A\ (ordermap\ A\ VWF)$   
**have**  $small\ A$   
**using**  $down\ that\ by\ auto$   
**then** **have**  $inj\text{-}g$ :  $inj\text{-on}\ ?g\ (elts\ \alpha)$   
**by** ( $meson\ inj\text{-on}\text{-}inv\text{-into}\ less\text{-eq}\text{-}V\text{-def}\ ordermap\text{-}surj\ ot\ subset\text{-}trans$ )  
**have**  $om\text{-}A\text{-}less$ :  $\bigwedge x\ y. \llbracket x \in A; y \in A; x < y \rrbracket \implies ordermap\ A\ VWF\ x <$   
 $ordermap\ A\ VWF\ y$   
**using**  $ord$   
**by** ( $meson\ A-\alpha\beta\ Ord\text{-in}\text{-}Ord\ VWF\text{-iff}\text{-}Ord\text{-}less\ \langle small\ A \rangle\ in\text{-}mono\ ordermap\text{-}mono\text{-}less\ trans\text{-}VWF\ wf\text{-}VWF$ )  
**have**  $\alpha\text{-}sub$ :  $elts\ \alpha \subseteq ordermap\ A\ VWF \text{' } A$   
**by** ( $metis\ \langle small\ A \rangle\ elts\text{-of}\text{-}set\ less\text{-eq}\text{-}V\text{-def}\ ordertype\text{-}def\ ot\ replacement$ )  
**have**  $g$ :  $?g \in elts\ \alpha \rightarrow elts\ (\alpha*\beta)$   
**by** ( $meson\ A-\alpha\beta\ Pi\text{-}I'\ \alpha\text{-}sub\ inv\text{-into}\text{-}into\ subset\text{-}eq$ )  
**then** **have**  $fg$ :  $f \circ (\lambda X. ?g \text{' } X) \in [elts\ \alpha]^2 \rightarrow \{..<2\}$   
**by** ( $rule\ nsets\text{-}compose\text{-}image\text{-}funcset\ [OF\ f\ \text{-}\ inj\text{-}g]$ )  
**obtain**  $i\ H$  **where**  $i < 2\ H \subseteq elts\ \alpha$   
**and**  $ot\text{-}eq$ :  $tp\ H = [k,\gamma]!i\ (f \circ (\lambda X. ?g \text{' } X)) \text{' } (nsets\ H\ 2) \subseteq \{i\}$   
**using**  $ii\ partn\text{-}lst\text{-}E\ [OF\ part\ fg]$  **by** ( $auto\ simp$ :  $eval\text{-}nat\text{-}numeral$ )  
**then** **consider**  $(0)\ i=0 \mid (1)\ i=1$   
**by**  $linarith$   
**then** **show**  $?thesis$   
**proof**  $cases$   
**case**  $0$   
**then** **have**  $f \text{' } [inv\text{-into}\ A\ (ordermap\ A\ VWF) \text{' } H]^2 \subseteq \{0\}$   
**using**  $ot\text{-}eq\ \langle H \subseteq elts\ \alpha \rangle\ \alpha\text{-}sub$  **by** ( $auto\ simp$ :  $nsets\text{-}def\ [of\ \text{-}\ k]$   
 $inj\text{-on}\text{-}inv\text{-into}\ elim!$ :  $nset\text{-}image\text{-}obtains$ )  
**moreover** **have**  $finite\ H \wedge card\ H = k$   
**using**  $0\ ot\text{-}eq\ \langle H \subseteq elts\ \alpha \rangle\ down$  **by** ( $simp\ add$ :  $finite\ ordertype\text{-}eq\text{-}card$ )  
**then** **have**  $inv\text{-into}\ A\ (ordermap\ A\ VWF) \text{' } H \in [A]^k$   
**using**  $\langle H \subseteq elts\ \alpha \rangle\ \alpha\text{-}sub$  **by** ( $auto\ simp$ :  $nsets\text{-}def\ [of\ \text{-}\ k]\ card\text{-}image$   
 $inj\text{-on}\text{-}inv\text{-into}\ inv\text{-into}\text{-}into$ )  
**ultimately** **show**  $?thesis$   
**by**  $blast$   
**next**  
**case**  $1$   
**have**  $gH$ :  $?g \text{' } H \subseteq elts\ (\alpha*\beta)$   
**by** ( $metis\ A-\alpha\beta\ \alpha\text{-}sub\ \langle H \subseteq elts\ \alpha \rangle\ image\text{-}subsetI\ inv\text{-into}\text{-}into\ subset\text{-}eq$ )  
**have**  $g\text{-}less$ :  $?g\ x < ?g\ y$  **if**  $x < y\ x \in elts\ \alpha\ y \in elts\ \alpha$  **for**  $x\ y$

**using** *Pi-mem* [*OF g*] *ord that*  
**by** (*meson*  $A\text{-}\alpha\beta$  *Ord-in-Ord Ord-not-le*  $\langle \text{small } A \rangle$  *dual-order.trans*  
*elts-subset-ON inv-ordermap-VWF-mono-le ot vsubsetD*)  
**have** [*simp*]:  $tp (?g \text{ ' } H) = tp H$   
**by** (*meson*  $\langle H \subseteq \text{elts } \alpha \rangle$  *ord down dual-order.trans elts-subset-ON gH*  
*g-less ordertype-VWF-inc-eq subsetD*)  
**show** *?thesis*  
**using** *ii* [*of ?g ' H*] *subset-inj-on* [*OF inj-g*  $\langle H \subseteq \text{elts } \alpha \rangle$ ] *ot-eq 1*  
**by** (*auto simp: gH elim!: nset-image-obtains*)  
**qed**  
**qed**  
**define** *K* **where**  $K \equiv \lambda i x. \{y \in \text{elts } (\alpha*\beta). x \neq y \wedge f\{x,y\} = i\}$   
**have** *small-K*: *small* ( $K i x$ ) **for**  $i x$   
**by** (*simp add: K-def*)  
**define** *KI* **where**  $KI \equiv \lambda i X. (\bigcap x \in X. K i x)$   
**have** *KI-disj-self*:  $X \cap KI i X = \{\}$  **for**  $i X$   
**by** (*auto simp: KI-def K-def*)  
**define** *M* **where**  $M \equiv \lambda D \mathfrak{A} x. \{\nu::V. \nu \in D \wedge tp (K 1 x \cap \mathfrak{A} \nu) \geq \alpha\}$   
**have** *M-sub-D*:  $M D \mathfrak{A} x \subseteq D$  **for**  $D \mathfrak{A} x$   
**by** (*auto simp: M-def*)  
**have** *small-M* [*simp*]: *small* ( $M D \mathfrak{A} x$ ) **if** *small D* **for**  $D \mathfrak{A} x$   
**by** (*simp add: M-def that*)  
**have** *9*:  $tp \{x \in A. tp (M D \mathfrak{A} x) \geq tp D\} \geq \alpha$  (**is** *ordertype ?AD -  $\geq \alpha$* )  
**if** *inD*: *indecomposable* ( $tp D$ ) **and** *D*:  $D \subseteq \text{elts } \beta$  **and** *A*:  $A \subseteq \text{elts } (\alpha*\beta)$   
**and** *tpA*:  $tp A = \alpha$   
**and**  $\mathfrak{A}$ :  $\mathfrak{A} \in D \rightarrow \{X. X \subseteq \text{elts } (\alpha*\beta) \wedge tp X = \alpha\}$  **for**  $D A \mathfrak{A}$   
— *remark (9), assuming an indecomposable order type*  
**proof** (*rule ccontr*)  
**define** *A'* **where**  $A' \equiv \{x \in A. \neg tp (M D \mathfrak{A} x) \geq tp D\}$   
**have** *small [iff]*: *small A small D*  
**using** *A D down* **by** (*auto simp: M-def*)  
**have** *small- $\mathfrak{A}$* : *small* ( $\mathfrak{A} \delta$ ) **if**  $\delta \in D$  **for**  $\delta$   
**using** *that  $\mathfrak{A}$*  **by** (*auto simp: Pi-iff subset-iff-less-eq-V*)  
**assume** *not- $\alpha$ -le*:  $\neg \alpha \leq tp \{x \in A. tp (M D \mathfrak{A} x) \geq tp D\}$   
**moreover**  
**obtain** *small A small A'*  $A' \subseteq A$  **and** *A'-sub*:  $A' \subseteq \text{elts } (\alpha*\beta)$   
**using** *A'-def A down* **by** *auto*  
**moreover** **have**  $A' = A - ?AD$   
**by** (*force simp: A'-def*)  
**ultimately** **have** *A'-ge*:  $tp A' \geq \alpha$   
**by** (*metis (no-types, lifting) dual-order.refl indec indecomposable-ordertype-eq*  
*mem-Collect-eq subsetI tpA*)  
**obtain** *X* **where**  $X \subseteq A'$  *finite X card X = k* **and** *fX0*:  $f \text{ ' } [X]^2 \subseteq \{0\}$   
**using** *Ak0* [*OF A'-sub A'-ge*] **by** (*auto simp: nsets-def [of - k]*)  
**then** **have**  $\ddagger$ :  $\neg tp (M D \mathfrak{A} x) \geq tp D$  **if**  $x \in X$  **for**  $x$   
**using** *that* **by** (*auto simp: A'-def*)  
**obtain** *x* **where**  $x \in X$   
**using**  $\langle \text{card } X = k \rangle \langle k > 1 \rangle$  **by** *fastforce*  
**have**  $\neg D \subseteq (\bigcup x \in X. M D \mathfrak{A} x)$

**proof**  
**assume**  $\text{not}: D \subseteq (\bigcup_{x \in X}. M D \mathfrak{A} x)$   
**have**  $\exists X \in M D \mathfrak{A} \langle X. \text{tp } D \leq \text{tp } X$   
**proof** (*rule indecomposable-ordertype-finite-ge [OF inD]*)  
**show**  $M D \mathfrak{A} \langle X \neq \{\}$   
**using**  $A'$ -def  $A'$ -ge not not- $\alpha$ -le **by** *auto*  
**show**  $\text{small} (\bigcup (M D \mathfrak{A} \langle X))$   
**using**  $\langle \text{finite } X \rangle$  **by** (*simp add: finite-imp-small*)  
**qed** (*use*  $\langle \text{finite } X \rangle$  **not in** *auto*)  
**then show** *False*  
**by** (*simp add: †*)  
**qed**  
**then obtain**  $\nu$  **where**  $\nu \in D$  **and**  $\nu: \nu \notin (\bigcup_{x \in X}. M D \mathfrak{A} x)$   
**by** *blast*  
**define**  $\mathcal{A}$  **where**  $\mathcal{A} \equiv \{KI 0 X \cap \mathfrak{A} \nu, \bigcup_{x \in X}. K 1 x \cap \mathfrak{A} \nu, X \cap \mathfrak{A} \nu\}$   
**have**  $\alpha\beta: X \subseteq \text{elts} (\alpha*\beta) \mathfrak{A} \nu \subseteq \text{elts} (\alpha*\beta)$   
**using**  $A'$ -sub  $\langle X \subseteq A' \rangle \mathfrak{A} \langle \nu \in D \rangle$  **by** *auto*  
**then have**  $KI 0 X \cup (\bigcup_{x \in X}. K 1 x) \cup X = \text{elts} (\alpha*\beta)$   
**using**  $\langle x \in X \rangle f$  **by** (*force simp: K-def KI-def Pi-iff less-2-cases-iff*)  
**with**  $\alpha\beta$  **have**  $\mathfrak{A}\nu\text{-}\mathcal{A}: \text{finite } \mathcal{A} \mathfrak{A} \nu \subseteq \bigcup \mathcal{A}$   
**by** (*auto simp: A-def*)  
**then have**  $\neg \text{tp} (K 1 x \cap \mathfrak{A} \nu) \geq \alpha$  **if**  $x \in X$  **for**  $x$   
**using** *that*  $\langle \nu \in D \rangle \nu \langle k > 1 \rangle \langle \text{card } X = k \rangle$  **by** (*fastforce simp: M-def*)  
**moreover have**  $\text{sm-K1}: \text{small} (\bigcup_{x \in X}. K 1 x \cap \mathfrak{A} \nu)$   
**by** (*meson Finite-V Int-lower2*  $\langle \nu \in D \rangle \langle \text{finite } X \rangle \text{small-}\mathfrak{A} \text{small-UN}$   
*smaller-than-small*)  
**ultimately have**  $\text{not1}: \neg \text{tp} (\bigcup_{x \in X}. K 1 x \cap \mathfrak{A} \nu) \geq \alpha$   
**using**  $\langle \text{finite } X \rangle \langle x \in X \rangle$  *indecomposable-ordertype-finite-ge [OF indec, of*  
 $(\lambda x. K 1 x \cap \mathfrak{A} \nu) \langle X \rangle$  **by** *blast*  
**moreover have**  $\neg \text{tp} (X \cap \mathfrak{A} \nu) \geq \alpha$   
**using**  $\langle \text{finite } X \rangle \langle \alpha \geq \omega \rangle$   
**by** (*meson finite-Int mem-not-refl ordertype-VWF- $\omega$  vsubsetD*)  
**moreover have**  $\alpha \leq \text{tp} (\mathfrak{A} \nu)$   
**using**  $\mathfrak{A} \langle \nu \in D \rangle \text{small-}\mathfrak{A}$  **by** *fastforce+*  
**moreover have**  $\text{small} (\bigcup \mathcal{A})$   
**using**  $\langle \nu \in D \rangle \text{small-}\mathfrak{A}$  **by** (*fastforce simp: A-def intro: smaller-than-small*  
*sm-K1*)  
**ultimately have**  $K0\mathfrak{A}\text{-ge}: \text{tp} (KI 0 X \cap \mathfrak{A} \nu) \geq \alpha$   
**using** *indecomposable-ordertype-finite-ge [OF indec  $\mathfrak{A}\nu\text{-}\mathcal{A}$ ]* **by** (*auto simp:*  
*A-def*)  
**have**  $\mathfrak{A}\nu: \mathfrak{A} \nu \subseteq \text{elts} (\alpha*\beta) \text{tp} (\mathfrak{A} \nu) = \alpha$   
**using**  $\langle \nu \in D \rangle \mathfrak{A}$  **by** *blast+*  
**then obtain**  $Y$  **where**  $Y_{\text{sub}}: Y \subseteq KI 0 X \cap \mathfrak{A} \nu$  **and** *finite*  $Y$   $\text{card } Y = k$   
**and**  $fY0: f \langle [Y]^2 \subseteq \{0\} \rangle$   
**using**  $Ak0$  [*OF - K0A-ge*] **by** (*auto simp: nsets-def [of - k]*)  
**have**  $\dagger: X \cap Y = \{\}$   
**using**  $Y_{\text{sub}}$  *KI-disj-self* **by** *blast*  
**then have**  $\text{card} (X \cup Y) = 2*k$   
**by** (*simp add:*  $\langle \text{card } X = k \rangle \langle \text{card } Y = k \rangle \langle \text{finite } X \rangle \langle \text{finite } Y \rangle$ )

*card-Un-disjoint*)  
**moreover have**  $X \cup Y \subseteq \text{elts } (\alpha * \beta)$   
**using**  $A'$ -sub  $\langle X \subseteq A' \rangle$   $\mathfrak{A} \nu(1)$   $\langle Y \subseteq KI\ 0\ X \cap \mathfrak{A}\ \nu \rangle$  **by** *auto*  
**moreover have**  $f '[X \cup Y]^2 \subseteq \{0\}$   
**using**  $fX0\ fY0\ Ysub$  **by** (*auto simp: † nsets-disjoint-2 image-Un image-UN*)  
*KI-def K-def*)  
**ultimately show** *False*  
**using**  $i$   $\langle finite\ X \rangle$   $\langle finite\ Y \rangle$  *ordertype-VWF-finite-nat* **by** *auto*  
**qed**  
**have**  $IX: tp\ \{x \in A.\ tp\ (M\ D\ \mathfrak{A}\ x) \geq tp\ D\} \geq \alpha$   
**if**  $D: D \subseteq \text{elts } \beta$  **and**  $A: A \subseteq \text{elts } (\alpha * \beta)$  **and**  $tpA: tp\ A = \alpha$   
**and**  $\mathfrak{A}: \mathfrak{A} \in D \rightarrow \{X.\ X \subseteq \text{elts } (\alpha * \beta) \wedge tp\ X = \alpha\}$  **for**  $D\ A\ \mathfrak{A}$   
— remark (9) for any order type  
**proof** –  
**obtain**  $L$  **where**  $UL: \bigcup (List.set\ L) \subseteq D$   
**and**  $indL: \bigwedge X.\ X \in List.set\ L \implies indecomposable\ (tp\ X)$   
**and**  $eqL: \bigwedge M.\ \llbracket M \subseteq D; \bigwedge X.\ X \in List.set\ L \implies tp\ (M \cap X) \geq tp\ X \rrbracket$   
 $\implies tp\ M = tp\ D$   
**using** *ord* **by** (*metis strong-ordertype-eq D order-refl*)  
**obtain**  $A''$  **where**  $A'': A'' \subseteq A\ tp\ A'' \geq \alpha$   
**and**  $\bigwedge x\ X.\ \llbracket x \in A''; X \in List.set\ L \rrbracket \implies tp\ (M\ D\ \mathfrak{A}\ x \cap X) \geq tp\ X$   
**using**  $UL\ indL$   
**proof** (*induction L arbitrary: thesis*)  
**case** (*Cons X L*)  
**then obtain**  $A''$  **where**  $A'': A'' \subseteq A\ tp\ A'' \geq \alpha$  **and**  $X \subseteq D$   
**and**  $ge-X: \bigwedge x\ X.\ \llbracket x \in A''; X \in List.set\ L \rrbracket \implies tp\ (M\ D\ \mathfrak{A}\ x \cap X) \geq tp\ X$   
 $X$  **by** *auto*  
**then have**  $tp-A'': tp\ A'' = \alpha$   
**by** (*metis A antisym down ordertype-VWF-mono tpA*)  
**have**  $ge-\alpha: tp\ \{x \in A''.\ tp\ (M\ X\ \mathfrak{A}\ x) \geq tp\ X\} \geq \alpha$   
**by** (*rule 9*) (*use A A'' tp-A'' Cons.premis*  $\langle D \subseteq \text{elts } \beta \rangle$   $\langle X \subseteq D \rangle$   $\mathfrak{A}$  **in**  
*auto*)  
**let**  $?A = \{x \in A''.\ tp\ (M\ D\ \mathfrak{A}\ x \cap X) \geq tp\ X\}$   
**have**  $M\text{-eq}: M\ D\ \mathfrak{A}\ x \cap X = M\ X\ \mathfrak{A}\ x$  **if**  $x \in A''$  **for**  $x$   
**using** *that*  $\langle X \subseteq D \rangle$  **by** (*auto simp: M-def*)  
**show** *thesis*  
**proof** (*rule Cons.premis*)  
**show**  $\alpha \leq tp\ ?A$   
**using**  $ge-\alpha$  **by** (*simp add: M-eq cong: conj-cong*)  
**show**  $tp\ Y \leq tp\ (M\ D\ \mathfrak{A}\ x \cap Y)$  **if**  $x \in ?A\ Y \in list.set\ (X\ \# L)$  **for**  $x\ Y$   
**using** *that*  $ge-X$  **by** *force*  
**qed** (*use A'' in auto*)  
**qed** (*use tpA in auto*)  
**then have**  $tp-M-ge: tp\ (M\ D\ \mathfrak{A}\ x) \geq tp\ D$  **if**  $x \in A''$  **for**  $x$   
**using**  $eqL$  *that* **by** (*auto simp: M-def*)  
**have**  $\alpha \leq tp\ A''$   
**by** (*simp add: A''*)  
**also have**  $\dots \leq tp\ \{x \in A''.\ tp\ (M\ D\ \mathfrak{A}\ x) \geq tp\ D\}$   
**by** (*metis (mono-tags, lifting) tp-M-ge eq-iff mem-Collect-eq subsetI*)

**also have**  $\dots \leq tp \{x \in A. tp D \leq tp (M D \mathfrak{A} x)\}$   
**by** (rule ordertype-mono) (use  $A'' A$  down in auto)  
**finally show** ?thesis .  
**qed**  
**have**  $IX'$ :  $tp \{x \in A'. tp (K 1 x \cap A) \geq \alpha\} \geq \alpha$   
**if**  $A: A \subseteq elts (\alpha*\beta)$   $tp A = \alpha$  **and**  $A': A' \subseteq elts (\alpha*\beta)$   $tp A' = \alpha$  **for**  $A A'$   
**proof** –  
**have**  $\ddagger: \alpha \leq tp (K 1 t \cap A)$  **if**  $1 \leq tp \{\nu. \nu = 0 \wedge \alpha \leq tp (K 1 t \cap A)\}$  **for**  
 $t$   
**using** that  
**by** (metis Collect-empty-eq less-eq-V-0-iff ordertype-empty zero-neq-one)  
**have**  $tp \{x \in A'. 1 \leq tp \{\nu. \nu = 0 \wedge \alpha \leq tp (K 1 x \cap A)\}\}$   
 $\leq tp \{x \in A'. \alpha \leq tp (K 1 x \cap A)\}$   
**by** (rule ordertype-mono) (use  $\ddagger A'$  in ‹auto simp: down›)  
**then show** ?thesis  
**using**  $IX$  [of  $\{0\} A' \lambda x. A$ ] that ‹ $0 \in elts \beta$ › **by** (force simp: M-def)  
**qed**  
**have**  $10: \exists x0 \in A. \exists g \in elts \beta \rightarrow elts \beta. strict-mono-on (elts \beta) g \wedge (\forall \nu \in F. g \nu = \nu)$   
 $\wedge (\forall \nu \in elts \beta. tp (K 1 x0 \cap \mathfrak{A} (g \nu)) \geq \alpha)$   
**if**  $F: finite F F \subseteq elts \beta$   
**and**  $A: A \subseteq elts (\alpha*\beta)$   $tp A = \alpha$   
**and**  $\mathfrak{A}: \mathfrak{A} \in elts \beta \rightarrow \{X. X \subseteq elts (\alpha*\beta) \wedge tp X = \alpha\}$   
**for**  $F A \mathfrak{A}$   
**proof** –  
**define**  $p$  where  $p \equiv card F$   
**have**  $\beta \notin F$   
**using** that **by** auto  
**then obtain**  $\iota :: nat \Rightarrow V$  **where**  $bij\iota: bij-betw \iota \{..p\}$  (insert  $\beta F$ ) **and**  
 $monu: strict-mono-on \{..p\} \iota$   
**using** ZFC-Cardinals.ex-bij-betw-strict-mono-card [of insert  $\beta F$ ]  $elts-subset-ON$   
‹ $Ord \beta$ ›  $F$   
**by** (simp add: p-def lessThan-Suc-atMost) blast  
**have**  $less-\iota-I: \iota k < \iota l$  **if**  $k < l l \leq p$  **for**  $k l$   
**using**  $monu$  that **by** (auto simp: strict-mono-on-def)  
**then have**  $less-\iota-D: k < l$  **if**  $\iota k < \iota l k \leq p$  **for**  $k l$   
**by** (metis less-asym linorder-neqE-nat that)  
**have**  $Ord-\iota: Ord (\iota k)$  **if**  $k \leq p$  **for**  $k$   
**by** (metis (no-types, lifting) ON-imp-Ord atMost-iff insert-subset mem-Collect-eq  
order-trans ‹ $F \subseteq elts \beta$ ›  $bij\iota$   $bij-betwE$   $elts-subset-ON$  ‹ $Ord \beta$ › that)  
**have**  $le-\iota 0$  [simp]:  $\bigwedge j. j \leq p \implies \iota 0 \leq \iota j$   
**by** (metis eq-refl leI le-0-eq less-\iota-I less-imp-le)  
**have**  $le-\iota: \iota i \leq \iota (j - Suc 0)$  **if**  $i < j j \leq p$  **for**  $i j$   
**proof** (cases  $i$ )  
**case** 0 **then show** ?thesis  
**using**  $le-\iota 0$  that **by** auto  
**next**  
**case** (Suc  $i'$ ) **then show** ?thesis



by (metis (no-types, opaque-lifting) Suc-pred le-less less-Suc-eq less-Suc-eq-0-disj  
less-ι-I not-less-eq that)

qed

have [simp]:  $\iota p = \beta$

proof –

obtain  $k$  where  $k: \iota k = \beta k \leq p$

by (meson atMost-iff bij $\iota$  bij-betw-iff-bijections insertI1)

then have  $k = p \vee k < p$

by linarith

then show ?thesis

using bij $\iota$  ord  $k$  that(2)

by (metis OrdmemD atMost-iff bij-betw-iff-bijections insert-iff leD less-ι-D  
order-refl subsetD)

qed

have  $F$ -imp-Ex:  $\exists k < p. \xi = \iota k$  if  $\xi \in F$  for  $\xi$

proof –

obtain  $k$  where  $k: k \leq p \xi = \iota k$

by (metis  $\langle \xi \in F \rangle$  atMost-iff bij $\iota$  bij-betw-def imageE insert-iff)

then show ?thesis

using  $\langle \beta \notin F \rangle \langle \iota p = \beta \rangle$  le-imp-less-or-eq that by blast

qed

have  $F$ -imp-ge:  $\xi \geq \iota 0$  if  $\xi \in F$  for  $\xi$

using  $F$ -imp-Ex [OF that] by (metis dual-order.order-iff-strict le0 less-ι-I)

define  $D$  where  $D \equiv \lambda k. (if\ k=0\ then\ \{..<\iota\ 0\}\ else\ \{\iota\ (k-1)<..<\iota\ k\}) \cap$   
elts  $\beta$

have  $D\beta: D\ k \subseteq$  elts  $\beta$  for  $k$

by (auto simp: D-def)

then have small-D [simp]: small (D  $k$ ) for  $k$

by (meson down)

have  $M$ -Int-D:  $M$  (elts  $\beta$ )  $\mathcal{A} x \cap D\ k = M$  (D  $k$ )  $\mathcal{A} x$  if  $k \leq p$  for  $x\ k$

using  $D\beta$  by (auto simp: M-def)

have  $\iota$ -le-if-D:  $\iota k \leq \mu$  if  $\mu \in D$  (Suc  $k$ ) for  $\mu\ k$

using that by (simp add: D-def order.order-iff-strict)

have mono-D:  $D\ i \ll D\ j$  if  $i < j\ j \leq p$  for  $i\ j$

proof (cases  $j$ )

case (Suc  $j'$ )

with that show ?thesis

apply (simp add: less-sets-def D-def Ball-def)

by (metis One-nat-def diff-Suc-1 le-ι less-le-trans less-trans)

qed (use that in auto)

then have disjnt-DD: disjnt (D  $i$ ) (D  $j$ ) if  $i \neq j\ i \leq p\ j \leq p$  for  $i\ j$

by (meson disjnt-sym less-linear less-sets-imp-disjnt that)

have UN-D-eq:  $(\bigcup l \leq k. D\ l) = \{..<\iota\ k\} \cap$  (elts  $\beta - F$ ) if  $k \leq p$  for  $k$

using that

proof (induction  $k$ )

case 0

then show ?case

```

    by (auto simp: D-def F-imp-ge leD)
  next
    case (Suc k)
    have D (Suc k)  $\cup$   $\{..<\iota k\} \cap (\text{elts } \beta - F) = \{..<\iota (Suc k)\} \cap (\text{elts } \beta - F)$ 
      (is ?lhs = ?rhs)
    proof
      show ?lhs  $\subseteq$  ?rhs
      using Suc.prem1
      by (auto simp: D-def if-split-mem2 intro: less- $\iota$ -I less-trans dest!: less- $\iota$ -D
        F-imp-Ex)
      have  $\bigwedge x. [x < \iota (Suc k); x \in \text{elts } \beta; x \notin F; \iota k \leq x] \implies \iota k < x$ 
        using Suc.prem1  $\langle F \subseteq \text{elts } \beta \rangle$  bij le-imp-less-or-eq
        by (fastforce simp: bij-betw-iff-bijections)
      then show ?rhs  $\subseteq$  ?lhs
        using Suc.prem1 by (auto simp: D-def Ord-not-less Ord-in-Ord [OF
           $\langle \text{Ord } \beta \rangle$ ] Ord- $\iota$  if-split-mem2)
      qed
    then
      show ?case
      using Suc by (simp add: atMost-Suc)
    qed
  have  $\beta$ -decomp:  $\text{elts } \beta = F \cup (\bigcup_{k \leq p} D k)$ 
    using  $\langle F \subseteq \text{elts } \beta \rangle$  OrdmemD [OF  $\langle \text{Ord } \beta \rangle$ ] by (auto simp: UN-D-eq)
  define  $\beta$ idx where  $\beta$ idx  $\equiv \lambda \nu. @k. \nu \in D k \wedge k \leq p$ 
  have  $\beta$ idx:  $\nu \in D (\beta$ idx  $\nu) \wedge \beta$ idx  $\nu \leq p$  if  $\nu \in \text{elts } \beta - F$  for  $\nu$ 
    using that by (force simp:  $\beta$ idx-def  $\beta$ -decomp intro: someI-ex del: conjI)
  have any-imp- $\beta$ idx:  $k = \beta$ idx  $\nu$  if  $\nu \in D k$   $k \leq p$  for  $k \nu$ 
  proof (rule ccontr)
    assume non:  $k \neq \beta$ idx  $\nu$ 
    have  $\nu \notin F$ 
      using that UN-D-eq by auto
    then show False
      using disjnt-DD [OF non] by (metis D $\beta$  Diff-iff  $\beta$ idx disjnt-iff subsetD
        that)
    qed
  have  $\exists A'. A' \subseteq A \wedge \text{tp } A' = \alpha \wedge (\forall x \in A'. F \subseteq M (\text{elts } \beta) \mathfrak{A} x)$ 
    using F
  proof induction
    case (insert  $\nu$  F)
    then obtain  $A'$  where  $A' \subseteq A$  and  $A': A' \subseteq \text{elts } (\alpha * \beta)$   $\text{tp } A' = \alpha$  and
      FN:  $\bigwedge x. x \in A' \implies F \subseteq M (\text{elts } \beta) \mathfrak{A} x$ 
      using A(1) by auto
    define  $A''$  where  $A'' \equiv \{x \in A'. \alpha \leq \text{tp } (K 1 x \cap \mathfrak{A} \nu)\}$ 
    have  $\nu \in \text{elts } \beta$   $F \subseteq \text{elts } \beta$ 
      using insert by auto
    note ordertype-eq-Ord [OF  $\langle \text{Ord } \beta \rangle$ , simp]
    show ?case
    proof (intro exI conjI)
      show  $A'' \subseteq A$ 

```

```

    using  $\langle A' \subseteq A \rangle$  by (auto simp:  $A''$ -def)
  show  $tp\ A'' = \alpha$ 
  proof (rule antisym)
    show  $tp\ A'' \leq \alpha$ 
      using  $\langle A'' \subseteq A \rangle$  down ordertype-VWF-mono  $A$  by blast
    have  $\mathfrak{A}\ \nu \subseteq elts\ (\alpha * \beta)$   $tp\ (\mathfrak{A}\ \nu) = \alpha$ 
      using  $\mathfrak{A}\ \langle \nu \in elts\ \beta \rangle$  by auto
    then show  $\alpha \leq tp\ A''$ 
      using  $IX'\ [OF\ -\ -\ A']$  by (simp add:  $A''$ -def)
  qed
  show  $\forall x \in A''.\ insert\ \nu\ F \subseteq M\ (elts\ \beta)\ \mathfrak{A}\ x$ 
    using  $A''$ -def  $FN\ M$ -def  $\langle \nu \in elts\ \beta \rangle$  by blast
  qed
  qed (use  $A$  in auto)
  then obtain  $A'$  where  $A': A' \subseteq A\ tp\ A' = \alpha$  and  $FN: \bigwedge x.\ x \in A' \implies F$ 
 $\subseteq M\ (elts\ \beta)\ \mathfrak{A}\ x$ 
    by metis
  have False
    if *:  $\bigwedge x0\ g.\ [\![x0 \in A; g \in elts\ \beta \rightarrow elts\ \beta; strict\ mono\ on\ (elts\ \beta)\ g]\!] \implies (\exists \nu \in F.\ g\ \nu \neq \nu) \vee (\exists \nu \in elts\ \beta.\ tp\ (K\ 1\ x0 \cap \mathfrak{A}\ (g\ \nu)) < \alpha)$ 
  proof -
    { fix  $x$  — construction of the monotone map  $g$  mentioned above
      assume  $x \in A'$ 
      with  $A'$  have  $x \in A$  by blast
      have  $\exists k.\ k \leq p \wedge tp\ (M\ (D\ k)\ \mathfrak{A}\ x) < tp\ (D\ k)$  (is ?P)
      proof (rule ccontr)
        assume  $\neg ?P$ 
        then have  $le: tp\ (D\ k) \leq tp\ (M\ (D\ k)\ \mathfrak{A}\ x)$  if  $k \leq p$  for  $k$ 
          by (meson Ord-linear2 Ord-ordertype that)
        have  $\exists f \in D\ k \rightarrow M\ (D\ k)\ \mathfrak{A}\ x.\ inj\ on\ f\ (D\ k) \wedge (strict\ mono\ on\ (D\ k)\ f)$ 
          if  $k \leq p$  for  $k$ 
            using le [OF that] that VWF-iff-Ord-less
            apply (clarsimp simp: ordertype-le-ordertype strict-mono-on-def)
            by (metis (full-types)  $D\beta\ M$ -sub- $D\ Ord$ -in- $Ord\ PiE\ VWF$ -iff-Ord-less
            ord(2) subsetD)
        then obtain  $h$  where  $fun\ h: \bigwedge k.\ k \leq p \implies h\ k \in D\ k \rightarrow M\ (D\ k)\ \mathfrak{A}\ x$ 
          and  $inj\ h: \bigwedge k.\ k \leq p \implies inj\ on\ (h\ k)\ (D\ k)$ 
          and  $mono\ h: \bigwedge k\ x\ y.\ k \leq p \implies strict\ mono\ on\ (D\ k)\ (h\ k)$ 
          by metis
        then have  $fun\ hD: \bigwedge k.\ k \leq p \implies h\ k \in D\ k \rightarrow D\ k$ 
          by (auto simp:  $M$ -def)
        have  $h$ -increasing:  $\nu \leq h\ k\ \nu$ 
          if  $k \leq p\ \nu \in D\ k$  for  $k\ \nu$ 
            by (meson  $D\beta\ Ord$ -mono-imp-increasing ord dual-order.trans
            elts-subset-ON  $fun\ hD\ mono\ h$  that)
        define  $g$  where  $g \equiv \lambda \nu.\ if\ \nu \in F\ then\ \nu\ else\ h\ (\beta id\ x\ \nu)\ \nu$ 
        have [simp]:  $g\ \nu = \nu$  if  $\nu \in F$  for  $\nu$ 
          using that by (auto simp:  $g$ -def)

```

```

have fun-g: g ∈ elts β → elts β
proof (rule Pi-I)
  fix x assume x ∈ elts β
  then have x ∈ D (βidx x) βidx x ≤ p if x ∉ F
    using that by (auto simp: βidx)
  then show g x ∈ elts β
    by (metis fun-h Dβ M-sub-D ⟨x ∈ elts β⟩ PiE g-def subsetD)
qed
have h-in-D: h (βidx ν) ν ∈ D (βidx ν) if ν ∉ F ν ∈ elts β for ν
  using βidx fun-hD that by fastforce
have 1: ι k < h (βidx ν) ν
  if k < p and ν: ν ∉ F ν ∈ elts β and ι k < ν for k ν
by (meson that h-in-D [OF ν] βidx DiffI h-increasing order-less-le-trans)
moreover have 2: h (βidx μ) μ < ι k
  if μ: μ ∉ F μ ∈ elts β and k < p μ < ι k for μ k
proof -
  have βidx μ ≤ k
  proof (rule ccontr)
    assume ¬ βidx μ ≤ k
    then have k < βidx μ
      by linarith
    then show False
      using ι-le-if-D βidx that by (metis Diff-iff Suc-pred le0 leD le-ι
le-less-trans)
  qed
  then show ?thesis
    using that h-in-D [OF μ]
  by (smt (verit, best) Int-lower1 UN-D-eq UN-I atMost-iff lessThan-iff
less-imp-le subset-eq)
qed
moreover have h (βidx μ) μ < h (βidx ν) ν
  if μ: μ ∉ F μ ∈ elts β and ν: ν ∉ F ν ∈ elts β and μ < ν for μ ν
proof -
  have le: βidx μ ≤ βidx ν if ι (βidx μ - Suc 0) < h (βidx μ) μ h
(βidx ν) ν < ι (βidx ν)
  by (metis 2 DiffI βidx μ ν ⟨μ < ν⟩ order.strict-trans h-increasing
leI le-ι order-less-asm order-less-le-trans that)
  have h 0 μ < h 0 ν if βidx μ = 0 βidx ν = 0
    using that mono-h unfolding strict-mono-on-def
  by (metis Diff-iff βidx μ ν ⟨μ < ν⟩)
  moreover have h 0 μ < h (βidx ν) ν
  if 0 < βidx ν h 0 μ < ι 0 and ι (βidx ν - Suc 0) < h (βidx ν) ν
  by (meson DiffI βidx ν le-ι le-less-trans less-le-not-le that)
  moreover have βidx ν ≠ 0
  if 0 < βidx μ h 0 ν < ι 0 ι (βidx μ - Suc 0) < h (βidx μ) μ
  using le le-0-eq that by fastforce
  moreover have h (βidx μ) μ < h (βidx ν) ν
  if ι (βidx μ - Suc 0) < h (βidx μ) μ h (βidx ν) ν < ι (βidx ν)
  h (βidx μ) μ < ι (βidx μ) ι (βidx ν - Suc 0) < h (βidx ν) ν

```

**using** *mono-h unfolding strict-mono-on-def*  
**by** (*metis le Diff-iff  $\beta \text{id}_x \mu \nu \langle \mu < \nu \rangle \text{le-}l \text{le-less le-less-trans}$  that*)  
**ultimately show** *?thesis*  
**using** *h-in-D [OF  $\mu$ ] h-in-D [OF  $\nu$ ] by (simp add: D-def split:*  
*if-split-asm)*  
**qed**  
**ultimately have** *sm-g: strict-mono-on (elts  $\beta$ ) g*  
**by** (*auto simp: g-def strict-mono-on-def dest!: F-imp-Ex*)  
**have** *False if  $\nu \in \text{elts } \beta$  and  $\nu: tp (K 1 x \cap \mathfrak{A} (g \nu)) < \alpha$  for  $\nu$*   
**proof** –  
**have**  *$F \subseteq M (\text{elts } \beta) \mathfrak{A} x$*   
**by** (*meson FN  $\langle x \in A' \rangle$* )  
**then have** *False if  $tp (K (Suc 0) x \cap \mathfrak{A} \nu) < \alpha \nu \in F$*   
**using** *that by (auto simp: M-def)*  
**moreover have** *False if  $tp (K (Suc 0) x \cap \mathfrak{A} (g \nu)) < \alpha \nu \in D k k$*   
 $\leq p \nu \notin F$  **for** *k*  
**proof** –  
**have**  *$h (\beta \text{id}_x \nu) \nu \in M (D (\beta \text{id}_x \nu)) \mathfrak{A} x$*   
**using** *fun-h  $\beta \text{id}_x \langle \nu \in \text{elts } \beta \rangle \langle \nu \notin F \rangle$  by auto*  
**then show** *False*  
**using** *that by (simp add: M-def g-def leD)*  
**qed**  
**ultimately show** *False*  
**using**  *$\langle \nu \in \text{elts } \beta \rangle \nu$  by (force simp:  $\beta$ -decomp)*  
**qed**  
**then show** *False*  
**using** *\* [OF  $\langle x \in A \rangle$  fun-g sm-g] by auto*  
**qed**  
**then have**  $\exists l. l \leq p \wedge tp (M (\text{elts } \beta) \mathfrak{A} x \cap D l) < tp (D l)$   
**using** *M-Int-D by auto*  
**}**  
**then obtain** *l where  $lp: \bigwedge x. x \in A' \implies l x \leq p$*   
**and** *lless:  $\bigwedge x. x \in A' \implies tp (M (\text{elts } \beta) \mathfrak{A} x \cap D (l x)) < tp (D (l x))$*   
**by** *metis*  
**obtain**  *$A'' L$  where  $A'' \subseteq A'$  and  $A'': A'' \subseteq \text{elts } (\alpha * \beta) tp A'' = \alpha$  and*  
 $lL: \bigwedge x. x \in A'' \implies l x = L$   
**proof** –  
**have** *eq:  $A' = (\bigcup_{i \leq p}. \{x \in A'. l x = i\})$*   
**using** *lp by auto*  
**have**  $\exists X \in (\lambda i. \{x \in A'. l x = i\}) ' \{..p\}. \alpha \leq tp X$   
**proof** (*rule indecomposable-ordertype-finite-ge [OF indec]*)  
**show** *small  $(\bigcup_{i \leq p}. \{x \in A'. l x = i\})$*   
**by** (*metis  $A'(1) A(1)$  eq down smaller-than-small*)  
**qed** (*use  $A'$  eq in auto*)  
**then show** *thesis*  
**proof**  
**fix**  *$A''$*   
**assume**  *$A'': A'' \in (\lambda i. \{x \in A'. l x = i\}) ' \{..p\}$  and  $\alpha \leq tp A''$*   
**then obtain**  *$L$  where  $L: \bigwedge x. x \in A'' \implies l x = L$*

```

    by auto
  have  $A'' \subseteq A'$ 
    using  $A''$  by force
  then have  $tp\ A'' \leq tp\ A'$ 
    by (meson  $A'\ A$  down order-trans ordertype-VWF-mono)
  with  $\langle \alpha \leq tp\ A'' \rangle$  have  $tp\ A'' = \alpha$ 
    using  $A'(2)$  by auto
  moreover have  $A'' \subseteq elts\ (\alpha * \beta)$ 
    using  $A'\ A\ \langle A'' \subseteq A' \rangle$  by auto
  ultimately show thesis
    using  $L$  that [OF  $\langle A'' \subseteq A' \rangle$ ] by blast
qed
qed
have  $\mathfrak{A}D: \mathfrak{A} \in D\ L \rightarrow \{X. X \subseteq elts\ (\alpha * \beta) \wedge tp\ X = \alpha\}$ 
  using  $\mathfrak{A}\ D\ \beta$  by blast
have  $\alpha: \alpha \leq tp\ \{x \in A''. tp\ (D\ L) \leq tp\ (M\ (D\ L)\ \mathfrak{A}\ x)\}$ 
  using IX [OF  $D\ \beta\ A''\ \mathfrak{A}D$ ] by simp
have  $M\ (elts\ \beta)\ \mathfrak{A}\ x \cap D\ L = M\ (D\ L)\ \mathfrak{A}\ x$  for  $x$ 
  using  $D\ \beta$  by (auto simp: M-def)
then have  $tp\ (M\ (D\ L)\ \mathfrak{A}\ x) < tp\ (D\ L)$  if  $x \in A''$  for  $x$ 
  using less that  $\langle A'' \subseteq A' \rangle$  lL by force
then have [simp]:  $\{x \in A''. tp\ (D\ L) \leq tp\ (M\ (D\ L)\ \mathfrak{A}\ x)\} = \{\}$ 
  using leD by blast
show False
  using  $\alpha\ \langle \alpha \geq \omega \rangle$  by simp
qed
then show ?thesis
  by (meson Ord-linear2 Ord-ordertype  $\langle Ord\ \alpha \rangle$ )
qed
let ?U = UNIV :: nat set
define  $\mu$  where  $\mu \equiv fst \circ from\ nat\ into\ (elts\ \beta \times ?U)$ 
define  $q$  where  $q \equiv to\ nat\ on\ (elts\ \beta \times ?U)$ 
have  $co\ \beta U: countable\ (elts\ \beta \times ?U)$ 
  by (simp add:  $\beta$  less- $\omega$ 1-imp-countable)
moreover have  $elts\ \beta \times ?U \neq \{\}$ 
  using  $\langle 0 \in elts\ \beta \rangle$  by blast
ultimately have  $range\ (from\ nat\ into\ (elts\ \beta \times ?U)) = (elts\ \beta \times ?U)$ 
  by (metis range-from-nat-into)
then have  $\mu\ in\ \beta$  [simp]:  $\mu\ i \in elts\ \beta$  for  $i$ 
  by (metis SigmaE  $\mu$ -def comp-apply fst-conv range-eqI)

then have  $Ord\ \mu$  [simp]:  $Ord\ (\mu\ i)$  for  $i$ 
  using Ord-in-Ord by blast

have  $inf\ \beta U: infinite\ (elts\ \beta \times ?U)$ 
  using  $\langle 0 \in elts\ \beta \rangle$  finite-cartesian-productD2 by auto

have 11 [simp]:  $\mu\ (q\ (\nu, n)) = \nu$  if  $\nu \in elts\ \beta$  for  $\nu\ n$ 
  by (simp add:  $\mu$ -def q-def that co- $\beta U$ )

```

**have** *range- $\mu$  [simp]*: *range  $\mu = \text{elts } \beta$*   
**by** (*auto simp: image-iff*) (*metis 11*)  
**have** [simp]: *KI i { } = UNIV KI i (insert a X) = K i a  $\cap$  KI i X* **for** *i a X*  
**by** (*auto simp: KI-def*)  
**define**  $\Phi$  **where**  $\Phi \equiv \lambda n::\text{nat}. \lambda \mathfrak{A} x. (\forall \nu \in \text{elts } \beta. \mathfrak{A} \nu \subseteq \text{elts } (\alpha * \beta) \wedge \text{tp } (\mathfrak{A} \nu) = \alpha)$   

$$\wedge x \text{ ' } \{..\lt n\} \subseteq \text{elts } (\alpha * \beta)$$
  

$$\wedge (\bigcup \nu \in \text{elts } \beta. \mathfrak{A} \nu) \subseteq \text{KI } 1 (x \text{ ' } \{..\lt n\})$$
  

$$\wedge \text{strict-mono-sets } (\text{elts } \beta) \mathfrak{A}$$
**define**  $\Psi$  **where**  $\Psi \equiv \lambda n::\text{nat}. \lambda g \mathfrak{A} \mathfrak{A}' xn. g \in \text{elts } \beta \rightarrow \text{elts } \beta \wedge \text{strict-mono-on } (\text{elts } \beta) g$   

$$\wedge (\forall i \leq n. g (\mu i) = \mu i)$$
  

$$\wedge (\forall \nu \in \text{elts } \beta. \mathfrak{A}' \nu \subseteq \text{K } 1 xn \cap \mathfrak{A} (g \nu))$$
  

$$\wedge \{xn\} \ll (\mathfrak{A}' (\mu n)) \wedge xn \in \mathfrak{A} (\mu n)$$
**let**  $\mathfrak{A}0 = \lambda \nu. \text{plus } (\alpha * \nu) \text{ ' } \text{elts } \alpha$   
**have** *base*:  $\Phi 0 \mathfrak{A}0 x$  **for**  $x$   
**by** (*auto simp:  $\Phi$ -def add-mult-less add-mult-less-add-mult ordertype-image-plus strict-mono-sets-def less-sets-def*)  
**have** *step*:  $Ex (\lambda (g, \mathfrak{A}', xn). \Psi n g \mathfrak{A} \mathfrak{A}' xn \wedge \Phi (\text{Suc } n) \mathfrak{A}' (x(n:=xn)))$  **if**  $\Phi n \mathfrak{A} x$  **for**  $n \mathfrak{A} x$   
**proof** –  
**have**  $\mathfrak{A}$ :  $\bigwedge \nu. \nu \in \text{elts } \beta \implies \mathfrak{A} \nu \subseteq \text{elts } (\alpha * \beta) \wedge \text{tp } (\mathfrak{A} \nu) = \alpha$   
**and**  $x$ :  $x \text{ ' } \{..\lt n\} \subseteq \text{elts } (\alpha * \beta)$   
**and** *sub*:  $\bigcup (\mathfrak{A} \text{ ' } \text{elts } \beta) \subseteq \text{KI } (\text{Suc } 0) (x \text{ ' } \{..\lt n\})$   
**and** *sm*: *strict-mono-sets* (*elts*  $\beta$ )  $\mathfrak{A}$   
**and**  $\mu\beta$ :  $\mu \text{ ' } \{..\lt n\} \subseteq \text{elts } \beta$  **and**  $\mathfrak{A}_{\text{sub}}$ :  $\mathfrak{A} (\mu n) \subseteq \text{elts } (\alpha * \beta)$   
**and**  $\mathfrak{A}_{\text{fun}}$ :  $\mathfrak{A} \in \text{elts } \beta \rightarrow \{X. X \subseteq \text{elts } (\alpha * \beta) \wedge \text{tp } X = \alpha\}$   
**using** *that* **by** (*auto simp:  $\Phi$ -def*)  
**then** **obtain**  $xn g$  **where**  $xn: xn \in \mathfrak{A} (\mu n)$  **and**  $g: g \in \text{elts } \beta \rightarrow \text{elts } \beta$   
**and** *sm-g*: *strict-mono-on* (*elts*  $\beta$ )  $g$  **and**  $g\text{-}\mu$ :  $\forall \nu \in \mu \text{ ' } \{..\lt n\}. g \nu = \nu$   
**and**  $g\text{-}\alpha$ :  $\forall \nu \in \text{elts } \beta. \alpha \leq \text{tp } (K 1 xn \cap \mathfrak{A} (g \nu))$   
**using** 10 [*OF* -  $\mu\beta$   $\mathfrak{A}_{\text{sub}}$  -  $\mathfrak{A}_{\text{fun}}$ ] **by** (*auto simp:  $\mathfrak{A}$* )  
**have** *tp1*:  $\text{tp } (K 1 xn \cap \mathfrak{A} (g \nu)) = \alpha$  **if**  $\nu \in \text{elts } \beta$  **for**  $\nu$   
**by** (*metis antisym Int-lower2 PiE  $\mathfrak{A}$  down g g- $\alpha$  ordertype-VWF-mono*)  
*that*)  
**have** *tp2*:  $\text{tp } (\mathfrak{A} (\mu n)) = \alpha$   
**by** (*auto simp:  $\mathfrak{A}$* )  
**obtain** *small* ( $\mathfrak{A} (\mu n)$ )  $\mathfrak{A} (\mu n) \subseteq ON$   
**by** (*meson  $\mathfrak{A}_{\text{sub}}$  ord down elts-subset-ON subset-trans*)  
**then** **obtain**  $A2$  **where**  $A2: \text{tp } A2 = \alpha \wedge A2 \subseteq K 1 xn \cap \mathfrak{A} (\mu n) \{xn\} \ll$   
 $A2$   
**using** *indecomposable-imp-Ex-less-sets* [*OF indec*  $\langle \alpha \geq \omega \rangle$  *tp2*]  
**by** (*metis  $\mu$ -in- $\beta$  atMost-iff image-eqI inf-le2 le-refl xn tp1 g- $\mu$* )  
**then** **have**  $A2\text{-sub}$ :  $A2 \subseteq \mathfrak{A} (\mu n)$  **by** *simp*  
**let**  $\mathfrak{A} = \lambda \nu. \text{if } \nu = \mu n \text{ then } A2 \text{ else } K 1 xn \cap \mathfrak{A} (g \nu)$   
**have** [simp]:  $(\{..\lt \text{Suc } n\} \cap \{x. x \neq n\}) = (\{..\lt n\})$   
**by** *auto*  
**have**  $K (\text{Suc } 0) xn \cap (\bigcup x \in \text{elts } \beta \cap \{\nu. \nu \neq \mu n\}. \mathfrak{A} (g x)) \subseteq \text{KI } (\text{Suc } 0)$   
 $(x \text{ ' } \{..\lt n\})$

**using** *sub g* **by** (*auto simp: KI-def*)  
**moreover have**  $A2 \subseteq KI (Suc\ 0) (x \text{ ' } \{..<n\}) A2 \subseteq elts (\alpha*\beta) xn \in elts$   
 $(\alpha*\beta)$   
**using**  $\mathfrak{A}_{sub}$  **sub**  $A2\ xn$  **by** *fastforce+*  
**moreover have** *strict-mono-sets* ( $elts\ \beta$ )  $\mathfrak{A}$   
**using** *sm sm-g g g- $\mu$  A2-sub*  
**unfolding** *strict-mono-sets-def strict-mono-on-def less-sets-def Pi-iff sub-*  
*set-iff Ball-def Bex-def image-iff*  
**by** (*simp (no-asm-use) add: if-split-mem2*) (*smt order-refl*)  
**ultimately have**  $\Phi (Suc\ n) \mathfrak{A} (x(n := xn))$   
**using** *tp1 x A2* **by** (*auto simp:  $\Phi$ -def K-def*)  
**with**  $A2$  **show** *?thesis*  
**by** (*rule-tac x=(g, $\mathfrak{A}$ ,xn) in exI*) (*simp add:  $\Psi$ -def g sm-g g- $\mu$  xn*)  
**qed**  
**define**  $G$  **where**  $G \equiv \lambda n\ \mathfrak{A}\ x.\ @ (g, \mathfrak{A}', x'). \exists xn. \Psi\ n\ g\ \mathfrak{A}\ \mathfrak{A}'\ xn \wedge x' =$   
 $(x(n:=xn)) \wedge \Phi (Suc\ n) \mathfrak{A}'\ x'$   
**have**  $G\Phi: (\lambda (g, \mathfrak{A}', x'). \Phi (Suc\ n) \mathfrak{A}'\ x') (G\ n\ \mathfrak{A}\ x)$   
**and**  $G\Psi: (\lambda (g, \mathfrak{A}', x'). \Psi\ n\ g\ \mathfrak{A}\ \mathfrak{A}'\ (x'\ n)) (G\ n\ \mathfrak{A}\ x)$  **if**  $\Phi\ n\ \mathfrak{A}\ x$  **for**  $n\ \mathfrak{A}\ x$   
**using** *step [OF that] by (force simp: G-def dest: some-eq-imp)+*  
**define**  $H$  **where**  $H \equiv rec\ nat\ (id, \mathfrak{A}0, undefined) (\lambda n\ (g0, \mathfrak{A}, x0). G\ n\ \mathfrak{A}\ x0)$   
**have**  $(\lambda (g, \mathfrak{A}, x). \Phi\ n\ \mathfrak{A}\ x) (H\ n)$  **for**  $n$   
**unfolding** *H-def* **by** (*induction n*) (*use G $\Phi$  base in fastforce*)  
**then have**  $H\text{-imp-}\Phi: \Phi\ n\ \mathfrak{A}\ x$  **if**  $H\ n = (g, \mathfrak{A}, x)$  **for**  $g\ \mathfrak{A}\ x\ n$   
**by** (*metis case-prodD that*)  
**then have**  $H\text{-imp-}\Psi: (\lambda (g, \mathfrak{A}', x'). let\ (g0, \mathfrak{A}, x) = H\ n\ in\ \Psi\ n\ g\ \mathfrak{A}\ \mathfrak{A}'\ (x'\ n))$   
 $(H (Suc\ n))$  **for**  $n$   
**using**  $G\Psi$  **by** (*fastforce simp: H-def split: prod.split*)  
**define**  $g$  **where**  $g \equiv \lambda n. (\lambda (g, \mathfrak{A}, x). g) (H (Suc\ n))$   
**have**  $g: g\ n \in elts\ \beta \rightarrow elts\ \beta$  **and** *sm-g: strict-mono-on* ( $elts\ \beta$ ) ( $g\ n$ )  
**and**  $13: \bigwedge i. i \leq n \implies g\ n\ (\mu\ i) = \mu\ i$  **for**  $n$   
**using**  $H\text{-imp-}\Psi$  [of  $n$ ] **by** (*auto simp: g-def  $\Psi$ -def*)  
**define**  $\mathfrak{A}$  **where**  $\mathfrak{A} \equiv \lambda n. (\lambda (g, \mathfrak{A}, x). \mathfrak{A}) (H\ n)$   
**define**  $x$  **where**  $x \equiv \lambda n. (\lambda (g, \mathfrak{A}, x). x\ n) (H (Suc\ n))$   
**have**  $14: \mathfrak{A} (Suc\ n)\ \nu \subseteq K\ 1\ (x\ n) \cap \mathfrak{A}\ n\ (g\ n\ \nu)$  **if**  $\nu \in elts\ \beta$  **for**  $\nu\ n$   
**using**  $H\text{-imp-}\Psi$  [of  $n$ ] **that** **by** (*force simp:  $\Psi$ -def  $\mathfrak{A}$ -def  $x$ -def  $g$ -def*)  
**then have**  $x14: \mathfrak{A} (Suc\ n)\ \nu \subseteq \mathfrak{A}\ n\ (g\ n\ \nu)$  **if**  $\nu \in elts\ \beta$  **for**  $\nu\ n$   
**using** *that* **by** *blast*  
**have**  $15: x\ n \in \mathfrak{A}\ n\ (\mu\ n)$  **and**  $16: \{x\ n\} \ll (\mathfrak{A} (Suc\ n)\ (\mu\ n))$  **for**  $n$   
**using**  $H\text{-imp-}\Psi$  [of  $n$ ] **by** (*force simp:  $\Psi$ -def  $\mathfrak{A}$ -def  $x$ -def*)  
**have**  $\mathfrak{A}\text{-}\alpha\beta: \mathfrak{A}\ n\ \nu \subseteq elts\ (\alpha*\beta)$  **if**  $\nu \in elts\ \beta$  **for**  $\nu\ n$   
**using**  $H\text{-imp-}\Phi$  [of  $n$ ] **that** **by** (*auto simp:  $\Phi$ -def  $\mathfrak{A}$ -def split: prod.split*)  
**have**  $12: strict-mono-sets$  ( $elts\ \beta$ ) ( $\mathfrak{A}\ n$ ) **for**  $n$   
**using**  $H\text{-imp-}\Phi$  [of  $n$ ] **that** **by** (*auto simp:  $\Phi$ -def  $\mathfrak{A}$ -def split: prod.split*)  
**let**  $?Z = range\ x$   
**have**  $S\text{-dec}: \bigcup (\mathfrak{A}\ (m+k) \text{ ' } elts\ \beta) \subseteq \bigcup (\mathfrak{A}\ m \text{ ' } elts\ \beta)$  **for**  $k\ m$   
**by** (*induction k*) (*use 14 g in <fastforce+>*)  
**have**  $x\ n \in K\ 1\ (x\ m)$  **if**  $m < n$  **for**  $m\ n$   
**proof** -  
**have**  $x\ n \in (\bigcup \nu \in elts\ \beta. \mathfrak{A}\ n\ \nu)$



by (*meson 15 UN-I  $\mu$ -in- $\beta$* )  
 also have  $\dots \subseteq (\bigcup \nu \in \text{elts } \beta. \mathfrak{A} (\text{Suc } m) \nu)$   
 using *S-dec [of Suc m] less-iff-Suc-add* that by *auto*  
 also have  $\dots \subseteq K 1 (x m)$   
 using *14* by *auto*  
 finally show *?thesis* .  
 qed  
 then have  $f\{x m, x n\} = 1$  if  $m < n$  for  $m n$   
 using that by (*auto simp: K-def*)  
 then have *Z-K1*:  $f ' [?Z]^2 \subseteq \{1\}$   
 by (*clarsimp simp: nsets-2-eq*) (*metis insert-commute less-linear*)  
 moreover have *Z-sub*:  $?Z \subseteq \text{elts } (\alpha * \beta)$   
 using *15  $\mathfrak{A}$ - $\alpha\beta$   $\mu$ -in- $\beta$*  by *blast*  
 moreover have *tp*  $?Z \geq \omega * \beta$   
 proof –  
 define **g** where  $\mathbf{g} \equiv \lambda i j x. \text{wfrec } (\lambda k. j - k) (\lambda \mathbf{g} k. \text{if } k < j \text{ then } \mathbf{g} k (\text{Suc } k) \text{ else } x) i$   
 have **g**:  $\mathbf{g} i j x = (\text{if } i < j \text{ then } \mathbf{g} i (\mathbf{g} (\text{Suc } i) j x) \text{ else } x)$  for  $i j x$   
 by (*simp add: g-def wfrec cut-apply*)  
 have *17*:  $\mathbf{g} k j (\mu i) = \mu i$  if  $i \leq k$  for  $i j k$   
 using *wf-measure [of  $\lambda k. j - k$ ]* that  
 by (*induction k rule: wf-induct-rule*) (*simp add: 13 g le-imp-less-Suc*)  
 have **g-in- $\beta$** :  $\mathbf{g} i j \nu \in \text{elts } \beta$  if  $\nu \in \text{elts } \beta$  for  $i j \nu$   
 using *wf-measure [of  $\lambda k. j - k$ ]* that  
 proof (*induction i rule: wf-induct-rule*)  
 case (*less i*)  
 with *g* show *?case* by (*force simp: g [of i]*)  
 qed  
 then have **g-fun**:  $\mathbf{g} i j \in \text{elts } \beta \rightarrow \text{elts } \beta$  for  $i j$   
 by *simp*  
 have **sm-g**: *strict-mono-on* ( $\text{elts } \beta$ ) ( $\mathbf{g} i j$ ) for  $i j$   
 using *wf-measure [of  $\lambda k. j - k$ ]*  
 proof (*induction i rule: wf-induct-rule*)  
 case (*less i*)  
 with *sm-g* show *?case*  
 by (*auto simp: g [of i] strict-mono-on-def g-in- $\beta$* )  
 qed  
 have \*:  $\mathfrak{A} j (\mu j) \subseteq \mathfrak{A} i (\mathbf{g} i j (\mu j))$  if  $i < j$  for  $i j$   
 using *wf-measure [of  $\lambda k. j - k$ ]* that  
 proof (*induction i rule: wf-induct-rule*)  
 case (*less i*)  
 then have  $j - \text{Suc } i < j - i$   
 by (*metis (no-types) Suc-diff-Suc lessI*)  
 with *less g-in- $\beta$*  show *?case*  
 by (*simp add: g [of i]*) (*metis 17 Suc-lessI  $\mu$ -in- $\beta$  order-refl order-trans*)  
 x14)

by (metis 17 Ord-in-Ord Ord-linear2  $\mu$ -in- $\beta$  leD le-refl less-V-def  $\langle$ Ord  $\beta$  $\rangle$ )  
 then have less-iff:  $\mathbf{g} \ i \ j \ (\mu \ j) < \mu \ i \longleftrightarrow \mu \ j < \mu \ i$  for  $i \ j$   
 by (metis (no-types, lifting) 17  $\mu$ -in- $\beta$  less-V-def order-refl sm-g strict-mono-on-def)  
 have eq-iff:  $\mathbf{g} \ i \ j \ (\mu \ j) = \mu \ i \longleftrightarrow \mu \ j = \mu \ i$  for  $i \ j$   
 by (metis eq-refl le-iff less-iff less-le)  
 have  $\mu$ -if-ne:  $\mu \ m < \mu \ n$  if  $mn$ :  $\mathfrak{A} \ m \ (\mu \ m) \ll \mathfrak{A} \ n \ (\mu \ n) \ m \neq n$  for  $m \ n$   
 proof –  
 have  $xmn$ :  $x \ m < x \ n$   
 using 15 less-setsD that(1) by blast  
 have Ordg: Ord (g n m ( $\mu \ m$ ))  
 using Ord-in-Ord g-in- $\beta$   $\mu$ -in- $\beta$  ord(2) by presburger  
 have  $\neg \mathfrak{A} \ m \ (\mu \ m) \ll \mathfrak{A} \ n \ (\mu \ n)$  if  $\mu \ n = \mu \ m$   
 using \* 15 eq-iff that unfolding less-sets-def  
 by (metis in-mono less-irrefl not-less-iff-gr-or-eq)  
 moreover  
 have  $\mathfrak{A} \ n \ (\mu \ n) \subseteq \mathfrak{A} \ m \ (\mathbf{g} \ m \ n \ (\mu \ n)) \vee \mathfrak{A} \ m \ (\mu \ m) \subseteq \mathfrak{A} \ n \ (\mathbf{g} \ n \ m \ (\mu \ m))$   
 using \* mn  
 by (meson antisym-conv3)  
 then have False if  $\mu \ n < \mu \ m$   
 using strict-mono-setsD [OF 12] 15  $xmn$  g-in- $\beta$   $\mu$ -in- $\beta$  that  
 by (smt (verit, best) Ordg Ord- $\mu$  Ord-linear2 leD le-iff less-asym less-iff  
 less-setsD subset-iff)  
 ultimately show  $\mu \ m < \mu \ n$   
 by (meson that(1) Ord- $\mu$  Ord-linear-lt)  
 qed  
 have 18:  $\mathfrak{A} \ m \ (\mu \ m) \ll \mathfrak{A} \ n \ (\mu \ n) \longleftrightarrow \mu \ m < \mu \ n$  for  $m \ n$   
 proof (cases  $n \ m$  rule: linorder-cases)  
 case less  
 show ?thesis  
 proof (intro iffI)  
 assume  $\mu \ m < \mu \ n$   
 then have  $\mathfrak{A} \ n \ (\mathbf{g} \ n \ m \ (\mu \ m)) \ll \mathfrak{A} \ n \ (\mu \ n)$   
 by (metis 12 g-in- $\beta$   $\mu$ -in- $\beta$  eq-iff le-iff less-V-def strict-mono-sets-def)  
 then show  $\mathfrak{A} \ m \ (\mu \ m) \ll \mathfrak{A} \ n \ (\mu \ n)$   
 by (meson \* less less-sets-weaken1)  
 qed (use  $\mu$ -if-ne less in blast)  
 next  
 case equal  
 with 15 show ?thesis by auto  
 next  
 case greater  
 show ?thesis  
 proof (intro iffI)  
 assume  $\mu \ m < \mu \ n$   
 then have  $\mathfrak{A} \ m \ (\mu \ m) \ll (\mathfrak{A} \ m \ (\mathbf{g} \ m \ n \ (\mu \ n)))$   
 by (meson 12 Ord-in-Ord Ord-linear2 g-in- $\beta$   $\mu$ -in- $\beta$  le-iff leD ord(2)  
 strict-mono-sets-def)  
 then show  $\mathfrak{A} \ m \ (\mu \ m) \ll \mathfrak{A} \ n \ (\mu \ n)$   
 by (meson \* greater less-sets-weaken2)

**qed** (use  $\mu$ -if-ne greater in blast)  
**qed**  
**have**  $\mathfrak{A}$ -increasing- $\mu$ :  $\mathfrak{A} n (\mu n) \subseteq \mathfrak{A} m (\mu m)$  **if**  $m \leq n$   $\mu m = \mu n$  **for**  $m n$   
**by** (metis \* 17 dual-order.order-iff-strict that)  
**moreover have**  $INF$ :  $infinite \{n. n \geq m \wedge \mu m = \mu n\}$  **for**  $m$   
**proof** –  
**have**  $infinite (range (\lambda n. q (\mu m, n)))$   
**unfolding**  $q$ -def  
**using**  $to$ -nat-on-infinite [ $OF$   $co$ - $\beta U$   $inf$ - $\beta U$ ]  $finite$ -image-iff  
**by** ( $simp$  add:  $finite$ -image-iff  $inj$ -on-def)  
**moreover have**  $(range (\lambda n. q (\mu m, n))) \subseteq \{n. \mu m = \mu n\}$   
**using** 11 [of  $\mu m$ ] **by**  $auto$   
**ultimately have**  $infinite \{n. \mu m = \mu n\}$   
**using**  $finite$ -subset **by**  $auto$   
**then have**  $infinite (\{n. \mu m = \mu n\} - \{..<m\})$   
**by**  $simp$   
**then show** ?thesis  
**by** ( $auto$   $simp$ :  $finite$ -nat-set-iff-bounded  $Bex$ -def not-less)  
**qed**  
**let**  $?eqv = \lambda m. \{n. m \leq n \wedge \mu m = \mu n\}$   
**have**  $sm$ - $x$ :  $strict$ -mono-on ( $?eqv m$ )  $x$  **for**  $m$   
**proof** ( $clarsimp$   $simp$ :  $strict$ -mono-on-def)  
**fix**  $n p$   
**assume**  $m \leq n$   $\mu p = \mu n$   $\mu m = \mu n$   $n < p$   
**with** 16 [of  $n$ ] **show**  $x n < x p$   
**by** (metis \* 15 17  $Suc$ -lessI  $insert$ -absorb  $insert$ -subset  $le$ - $SucI$   $less$ -sets-singleton1)  
**qed**  
**then have**  $inj$ - $x$ :  $inj$ -on  $x$  ( $?eqv m$ ) **for**  $m$   
**using**  $strict$ -mono-on-imp- $inj$ -on **by**  $blast$   
**define**  $ZA$  **where**  $ZA \equiv \lambda m. ?Z \cap \mathfrak{A} m (\mu m)$   
**have**  $small$ - $ZA$  [ $simp$ ]:  $small (ZA m)$  **for**  $m$   
**by** (metis  $ZA$ -def  $inf$ -le1  $small$ -image-nat  $smaller$ -than- $small$ )  
**have** 19:  $tp (ZA m) \geq \omega$  **for**  $m$   
**proof** –  
**have**  $x ' \{n. m \leq n \wedge \mu m = \mu n\} \subseteq ZA m$   
**unfolding**  $ZA$ -def **using** 15  $\mathfrak{A}$ -increasing- $\mu$  **by**  $blast$   
**then have**  $infinite (ZA m)$   
**using**  $INF$  [of  $m$ ]  $finite$ -image-iff [ $OF$   $inj$ - $x$ ] **by** ( $meson$   $finite$ -subset)  
**then show** ?thesis  
**by** ( $simp$  add:  $ordertype$ -infinite-ge- $\omega$ )  
**qed**  
**have**  $\exists f \in elts \omega \rightarrow ZA m. strict$ -mono-on ( $elts \omega$ )  $f$  **for**  $m$   
**proof** –  
**obtain**  $Z$  **where**  $Z \subseteq ZA m$   $tp Z = \omega$   
**by** ( $meson$  19  $Ord$ - $\omega$   $le$ -ordertype-obtains-subset  $small$ - $ZA$ )  
**moreover have**  $ZA m \subseteq ON$   
**using**  $Ord$ -in- $Ord$   $\mathfrak{A}$ - $\alpha\beta$   $\mu$ -in- $\beta$  **unfolding**  $ZA$ -def **by**  $blast$   
**ultimately show** ?thesis  
**by** (metis  $strict$ -mono-on-ordertype  $Pi$ -mono  $small$ - $ZA$   $smaller$ -than- $small$ )

subset-iff)  
**qed**  
**then obtain**  $\varphi$  **where**  $\varphi: \bigwedge m. \varphi m \in \text{elts } \omega \rightarrow ZA m$   
**and**  $sm\text{-}\varphi: \bigwedge m. \text{strict-mono-on } (\text{elts } \omega) (\varphi m)$   
**by** *metis*  
**have**  $Ex(\lambda(m,\nu). \nu \in \text{elts } \beta \wedge \gamma = \omega * \nu + \text{ord-of-nat } m)$  **if**  $\gamma \in \text{elts } (\omega * \beta)$  **for**  $\gamma$   
**using** *that* **by** (*auto simp: mult [of  $\omega \beta$ ] lift-def elts- $\omega$* )  
**then obtain** *split* **where**  $split: \bigwedge \gamma. \gamma \in \text{elts } (\omega * \beta) \implies$   
 $(\lambda(m,\nu). \nu \in \text{elts } \beta \wedge \gamma = \omega * \nu + \text{ord-of-nat } m)(split \gamma)$   
**by** *meson*  
**have** *split-eq [simp]: split*  $(\omega * \nu + \text{ord-of-nat } m) = (m,\nu)$  **if**  $\nu \in \text{elts } \beta$  **for**  
 $\nu m$   
**proof** –  
**have** [*simp*]:  $\omega * \nu + \text{ord-of-nat } m = \omega * \xi + \text{ord-of-nat } n \iff \xi = \nu \wedge$   
 $n = m$  **if**  $\xi \in \text{elts } \beta$  **for**  $\xi n$   
**by** (*metis Ord- $\omega$  that Ord-mem-iff-less-TC mult-cancellation-lemma ord-of-nat- $\omega$  ord-of-nat-inject*)  
**show** *?thesis*  
**using** *split [of  $\omega * \nu + m$ ] that* **by** (*auto simp: mult [of  $\omega \beta$ ] lift-def cong: conj-cong*)  
**qed**  
**define**  $\pi$  **where**  $\pi \equiv \lambda \gamma. (\lambda(m,\nu). \varphi (q(\nu,0)) m)(split \gamma)$   
**have**  $\pi\text{-Pi}: \pi \in \text{elts } (\omega * \beta) \rightarrow (\bigcup m. ZA m)$   
**using**  $\varphi$  **by** (*fastforce simp:  $\pi$ -def mult [of  $\omega \beta$ ] lift-def elts- $\omega$* )  
**moreover** **have**  $(\bigcup m. ZA m) \subseteq ON$   
**unfolding** *ZA-def* **using**  $\mathfrak{A}\text{-}\alpha\beta \mu\text{-in-}\beta \text{elts-subset-ON}$  **by** *blast*  
**ultimately** **have**  $Ord\text{-}\pi\text{-Pi}: \pi \in \text{elts } (\omega * \beta) \rightarrow ON$   
**by** *fastforce*  
**show** *tp ?Z  $\geq \omega * \beta$*   
**proof** –  
**have**  $\dagger: (\bigcup m. ZA m) = ?Z$   
**using** 15 **by** (*force simp: ZA-def*)  
**moreover**  
**have**  $tp (\text{elts } (\omega * \beta)) \leq tp (\bigcup m. ZA m)$   
**proof** (*rule ordertype-inc-le*)  
**show**  $\pi \text{ ‘ elts } (\omega * \beta) \subseteq (\bigcup m. ZA m)$   
**using**  $\pi\text{-Pi}$  **by** *blast*  
**next**  
**fix**  $u v$   
**assume**  $x: u \in \text{elts } (\omega * \beta)$  **and**  $y: v \in \text{elts } (\omega * \beta)$  **and**  $(u,v) \in VWF$   
**then** **have**  $u < v$   
**by** (*meson Ord- $\omega$  Ord-in-Ord Ord-mult VWF-iff-Ord-less ord(2)*)  
**moreover**  
**obtain**  $m \nu n \xi$  **where**  $ueq: u = \omega * \nu + \text{ord-of-nat } m$  **and**  $\nu: \nu \in \text{elts } \beta$   
**and**  $veq: v = \omega * \xi + \text{ord-of-nat } n$  **and**  $\xi: \xi \in \text{elts } \beta$   
**using**  $x y$  **by** (*auto simp: mult [of  $\omega \beta$ ] lift-def elts- $\omega$* )  
**ultimately** **have**  $\nu \leq \xi$   
**by** (*meson Ord- $\omega$  Ord-in-Ord Ord-linear2  $\langle Ord \beta \rangle$  add-mult-less-add-mult*)

```

less-asym ord-of-nat- $\omega$ 
  consider (eq)  $\nu = \xi \mid$  (lt)  $\nu < \xi$ 
    using  $\langle \nu \leq \xi \rangle$  le-neq-trans by blast
  then have  $\pi u < \pi v$ 
  proof cases
    case eq
      then have  $m < n$ 
        using ueq veq  $\langle u < v \rangle$  by simp
      then have  $\varphi (q (\xi, 0)) m < \varphi (q (\xi, 0)) n$ 
        using sm- $\varphi$  strict-mono-onD by blast
      then show ?thesis
        using eq ueq veq  $\nu \langle m < n \rangle$  by (simp add:  $\pi$ -def)
    next
      case lt
        have  $\varphi (q(\nu, 0)) m \in \mathfrak{A} (q(\nu, 0)) (\mu(q(\nu, 0))) \varphi (q (\xi, 0)) n \in \mathfrak{A} (q(\xi, 0))$ 
          ( $\mu(q(\xi, 0))$ )
          using  $\varphi$  unfolding ZA-def by blast+
        then show ?thesis
          using lt ueq veq  $\nu \xi$  18 [of  $q(\nu, 0)$   $q(\xi, 0)$ ]
          by (simp add:  $\pi$ -def less-sets-def)
        qed
        then show  $(\pi u, \pi v) \in VWF$ 
          using  $\pi$ -Pi by (metis Ord- $\pi$ -Pi PiE VWF-iff-Ord-less x y mem-Collect-eq)
        qed (use  $\dagger$  in auto)
        ultimately show ?thesis by simp
        qed
        then obtain  $Z$  where  $Z \subseteq ?Z$  tp  $Z = \omega * \beta$ 
          by (meson Ord- $\omega$  Ord-mult ord Z-sub down le-ordertype-obtains-subset)
        ultimately show False
          using iii [of  $Z$ ] by (meson dual-order.trans image-mono nsets-mono)
        qed
        have False
          if 0:  $\forall H. \text{tp } H = \text{ord-of-nat } (2*k) \longrightarrow H \subseteq \text{elts } (\alpha*\beta) \longrightarrow \neg f ' [H]^2 \subseteq \{0\}$ 
          and 1:  $\forall H. \text{tp } H = \text{min } \gamma (\omega * \beta) \longrightarrow H \subseteq \text{elts } (\alpha*\beta) \longrightarrow \neg f ' [H]^2 \subseteq$ 
          {1}
          proof (cases  $\omega*\beta \leq \gamma$ )
            case True
              then have  $\dagger: \exists H' \subseteq H. \text{tp } H' = \omega * \beta$  if tp  $H = \gamma$  small  $H$  for  $H$ 
                by (metis Ord- $\omega$  Ord- $\omega$ 1 Ord-in-Ord Ord-mult  $\beta$  le-ordertype-obtains-subset
                that)
              have [simp]:  $\text{min } \gamma (\omega*\beta) = \omega*\beta$ 
                by (simp add: min-absorb2 that True)
              then show ?thesis
                using  $*$  [OF 0] 1 True
                by simp (meson  $\dagger$  down image-mono nsets-mono subset-trans)
            next
              case False
                then have  $\dagger: \exists H' \subseteq H. \text{tp } H' = \gamma$  if tp  $H = \omega * \beta$  small  $H$  for  $H$ 

```

by (metis Ord-linear-le Ord-ordertype ‹Ord  $\gamma$ › le-ordertype-obtains-subset  
 that)  
 then have  $\gamma \leq \omega * \beta$   
 by (meson Ord- $\omega$  Ord- $\omega 1$  Ord-in-Ord Ord-linear-le Ord-mult  $\beta$  ‹Ord  $\gamma$ ›  
 False)  
 then have [simp]:  $\min \gamma (\omega * \beta) = \gamma$   
 by (simp add: min-absorb1)  
 then show ?thesis  
 using \* [OF 0] 1 False  
 by simp (meson † down image-mono nsets-mono subset-trans)  
 qed  
 then show  $\exists i < \text{Suc } (\text{Suc } 0). \exists H \subseteq \text{elts } (\alpha * \beta). \text{tp } H = [\text{ord-of-nat } (2 * k), \min$   
 $\gamma (\omega * \beta)] ! i \wedge f ' [H]^2 \subseteq \{i\}$   
 by force  
 qed  
 qed

**theorem Erdos-Milner:**

assumes  $\nu: \nu \in \text{elts } \omega 1$   
 shows partn-lst-VWF ( $\omega \uparrow (1 + \nu * n)$ ) [ord-of-nat ( $2 \hat{n}$ ),  $\omega \uparrow (1 + \nu)$ ] 2  
 proof (induction n)  
 case 0  
 then show ?case  
 using partn-lst-VWF-degenerate [of 1 2] by simp  
 next  
 case (Suc n)  
 have Ord  $\nu$   
 using Ord- $\omega 1$  Ord-in-Ord assms by blast  
 have  $1 + \nu \leq \nu + 1$   
 by (simp add: ‹Ord  $\nu$ › one-V-def plus-Ord-le)  
 then have [simp]:  $\min (\omega \uparrow (1 + \nu)) (\omega * \omega \uparrow \nu) = \omega \uparrow (1 + \nu)$   
 by (simp add: ‹Ord  $\nu$ › oexp-add min-def)  
 have ind: indecomposable ( $\omega \uparrow (1 + \nu * \text{ord-of-nat } n)$ )  
 by (simp add: ‹Ord  $\nu$ › indecomposable- $\omega$ -power)  
 show ?case  
 proof (cases n = 0)  
 case True  
 then show ?thesis  
 using partn-lst-VWF- $\omega$ -2 ‹Ord  $\nu$ › one-V-def by auto  
 next  
 case False  
 then have  $\text{Suc } 0 < 2 \hat{n}$   
 using less-2-cases not-less-eq by fastforce  
 then have partn-lst-VWF ( $\omega \uparrow (1 + \nu * n) * \omega \uparrow \nu$ ) [ord-of-nat ( $2 * 2 \hat{n}$ ),  
 $\omega \uparrow (1 + \nu)$ ] 2  
 using Erdos-Milner-aux [OF Suc ind, where  $\beta = \omega \uparrow \nu$ ] ‹Ord  $\nu$ ›  $\nu$   
 by (auto simp: countable-oexp)  
 then show ?thesis

```

    using ⟨Ord ν⟩ by (simp add: mult-succ mult.assoc oexp-add)
  qed
qed

```

```

corollary remark-3: partn-lst-VWF (ω↑(Suc(4*k))) [4, ω↑(Suc(2*k))] 2
  using Erdos-Milner [of 2*k 2]
  apply (simp flip: ord-of-nat-mult ord-of-nat.simps)
  by (simp add: one-V-def)

```

Theorem 3.2 of Jean A. Larson, *ibid.*

**corollary** *Theorem-3-2:*

```

  fixes k n::nat
  shows partn-lst-VWF (ω↑(n*k)) [ω↑n, ord-of-nat k] 2
proof (cases n=0 ∨ k=0)
  case True
  then show ?thesis
    by (auto intro: partn-lst-triv0 [where i=1] partn-lst-triv1 [where i=0])
  next
  case False
  then have n > 0 k > 0
    by auto
  from ⟨k > 0⟩ less-exp [of ⟨k - 1⟩] have ⟨k ≤ 2 ^ (k - 1)⟩
    by (cases k) (simp-all add: less-eq-Suc-le)
  have PV: partn-lst-VWF (ω ↑ (1 + ord-of-nat (n-1) * ord-of-nat (k-1)))
    [ord-of-nat (2 ^ (k-1)), ω ↑ (1 + ord-of-nat (n-1))] 2
    using Erdos-Milner [of ord-of-nat (n-1) k-1] Ord-ω1 Ord-mem-iff-lt less-imp-le
  by blast
  have k+n ≤ Suc (k * n)
    using False not0-implies-Suc by fastforce
  then have 1 + (n - 1) * (k - 1) ≤ n*k
    using False by (auto simp: algebra-simps)
  then have (1 + ord-of-nat (n - 1) * ord-of-nat (k - 1)) ≤ ord-of-nat(n*k)
    by (metis (mono-tags, lifting) One-nat-def one-V-def ord-of-nat.simps ord-of-nat-add
    ord-of-nat-mono-iff ord-of-nat-mult)
  then have x: ω ↑ (1 + ord-of-nat (n - 1) * ord-of-nat (k - 1)) ≤ ω↑(n*k)
    by (simp add: oexp-mono-le)
  then have partn-lst-VWF (ω↑(n*k)) [ω ↑ (1 + ord-of-nat (n-1)), ord-of-nat (2
  ^ (k-1))] 2
    by (metis PV partn-lst-two-swap Partitions.partn-lst-greater-resource less-eq-V-def)
  then have partn-lst-VWF (ω↑(n*k)) [ω ↑ n, ord-of-nat (2 ^ (k-1))] 2
    using ord-of-minus-1 [OF ⟨n > 0⟩] by (simp add: one-V-def)
  then show ?thesis
    using ⟨k ≤ 2 ^ (k - 1)⟩
    by (auto elim!: partn-lst-less simp add: less-Suc-eq)
qed
end

```

### 3 An ordinal partition theorem by Jean A. Larson

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

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

**begin**

**abbreviation** *list-of*  $\equiv$  *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-of-nat* m) + *omega-sum-aux* n ms

**abbreviation** *omega-sum* **where** *omega-sum* ms  $\equiv$  *omega-sum-aux* (length ms) ms

A normal expansion has no leading zeroes

**inductive** *normal*:: nat list  $\Rightarrow$  bool **where**  
*normal-Nil*[*iff*]: *normal* []  
| *normal-Cons*:  $m > 0 \implies \text{normal } (m\#ms)$

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

**lemma** *omega-sum-0-iff* [*simp*]: *normal* ns  $\implies$  *omega-sum* ns = 0  $\iff$  ns = []  
**by** (*induction* ns rule: *normal.induct*) *auto*

**lemma** *Ord-omega-sum-aux* [*simp*]: *Ord* (*omega-sum-aux* k ms)  
**by** (*induction* rule: *omega-sum-aux.induct*) *auto*

**lemma** *Ord-omega-sum*: *Ord* (*omega-sum* ms)  
**by** *simp*

**lemma** *omega-sum-less- $\omega\omega$*  [*intro*]: *omega-sum* ms  $<$   $\omega \uparrow \omega$

**proof** (*induction* ms)

**case** (*Cons* m ms)

**have**  $\omega \uparrow (\text{length } ms) * \text{ord-of-nat } m \in \text{elts } (\omega \uparrow \text{Suc } (\text{length } ms))$

**using** *Ord-mem-iff-lt* **by** *auto*

**then have**  $\omega \uparrow (\text{length } ms) * \text{ord-of-nat } m \in \text{elts } (\omega \uparrow \omega)$

**using** *Ord-ord-of-nat oexp-mono-le omega-nonzero ord-of-nat-le-omega* **by** *blast*



**with** *Cons* **show** ?*case*  
**by** (*auto simp: mult-succ OrdmemD oexp-less indecomposableD indecomposable- $\omega$ -power*)  
**qed** (*auto simp: zero-less-Limit*)

**lemma** *omega-sum-aux-less: omega-sum-aux k ms <  $\omega \uparrow k$*   
**proof** (*induction rule: omega-sum-aux.induct*)  
**case** ( $\exists n m ms$ )  
**have**  $\omega \uparrow n * \text{ord-of-nat } m + \omega \uparrow n < \omega \uparrow n * \omega$   
**by** (*metis Ord-ord-of-nat  $\omega$ -power-succ-gtr mult-succ oexp-succ ord-of-nat.simps(2)*)  
**with**  $\exists$  **show** ?*case*  
**using** *dual-order.strict-trans* **by** *force*  
**qed** *auto*

**lemma** *omega-sum-less: omega-sum ms <  $\omega \uparrow (\text{length } ms)$*   
**by** (*rule omega-sum-aux-less*)

**lemma** *omega-sum-ge: m  $\neq$  0  $\implies$   $\omega \uparrow (\text{length } ms) \leq \text{omega-sum } (m\#ms)$*   
**apply** *clarsimp*  
**by** (*metis Ord-ord-of-nat add-le-cancel-left0 le-mult Nat.neq0-conv ord-of-eq-0-iff vsubsetD*)

**lemma** *omega-sum-length-less:*  
**assumes** *normal ns length ms < length ns*  
**shows** *omega-sum ms < omega-sum ns*  
**using** *assms*  
**proof** (*induction rule: normal.induct*)  
**case** (*normal-Cons n ns'*)  
**have**  $\omega \uparrow \text{length } ms \leq \omega \uparrow \text{length } ns'$   
**using** *normal-Cons oexp-mono-le* **by** *auto*  
**then show** ?*case*  
**by** (*metis gr-implies-not-zero less-le-trans normal-Cons.hyps omega-sum-aux-less omega-sum-ge*)  
**qed** *auto*

**lemma** *omega-sum-length-leD:*  
**assumes** *omega-sum ms  $\leq$  omega-sum ns normal ms*  
**shows** *length ms  $\leq$  length ns*  
**by** (*meson assms leD leI omega-sum-length-less*)

**lemma** *omega-sum-less-eqlen-iff-cases [simp]:*  
**assumes** *length ms = length ns*  
**shows** *omega-sum (m#ms) < omega-sum (n#ns)  $\iff$  m < n  $\vee$  m = n  $\wedge$  omega-sum ms < omega-sum ns*  
**using** *omega-sum-less [of ms] omega-sum-less [of ns]*  
**by** (*metis Kirby.add-less-cancel-left Omega-Omega.Cons Ord- $\omega$  Ord-oexp Ord-omega-sum Ord-ord-of-nat assms length-Cons linorder-neqE-nat mult-nat-less-add-less order-less-asymp*)

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

**assumes**  $m > 0 \ n > 0$   
**shows**  $\text{omega-sum } (m\#ms) < \text{omega-sum } (n\#ns)$   
 $\longleftrightarrow \text{length } ms < \text{length } ns$   
 $\vee \text{length } ms = \text{length } ns \wedge m < n$   
 $\vee \text{length } ms = \text{length } ns \wedge m = n \wedge \text{omega-sum } ms < \text{omega-sum } ns$   
**by** (*smt (verit) assms length-Cons less-eq-Suc-le less-le-not-le nat-less-le normal-Cons not-le*  
*omega-sum-length-less omega-sum-less-eqlen-iff-cases*)

**lemma** *omega-sum-less-iff*:  
 $((\text{length } ms, \text{omega-sum } ms), (\text{length } ns, \text{omega-sum } ns)) \in \text{less-than } \langle *lex* \rangle$   
*VWF*  
 $\longleftrightarrow (ms, ns) \in \text{lenlex less-than}$   
**proof** (*induction ms arbitrary: ns*)  
**case** (*Cons m ms*)  
**then show** *?case*  
**proof** (*induction ns*)  
**case** (*Cons n ns'*)  
**show** *?case*  
**using** *Cons.premis Cons-lenlex-iff omega-sum-less-eqlen-iff-cases* **by** *fastforce*  
**qed** *auto*  
**qed** *auto*

**lemma** *eq-omega-sum-less-iff*:  
**assumes**  $\text{length } ms = \text{length } ns$   
**shows**  $(\text{omega-sum } ms, \text{omega-sum } ns) \in \text{VWF} \longleftrightarrow (ms, ns) \in \text{lenlex less-than}$   
**by** (*metis assms in-lex-prod less-not-refl less-than-iff omega-sum-less-iff*)

**lemma** *eq-omega-sum-eq-iff*:  
**assumes**  $\text{length } ms = \text{length } ns$   
**shows**  $\text{omega-sum } ms = \text{omega-sum } ns \longleftrightarrow ms = ns$   
**proof**  
**assume**  $\text{omega-sum } ms = \text{omega-sum } ns$   
**then have**  $(\text{omega-sum } ms, \text{omega-sum } ns) \notin \text{VWF } (\text{omega-sum } ns, \text{omega-sum } ms) \notin \text{VWF}$   
**by** *auto*  
**then obtain**  $(ms, ns) \notin \text{lenlex less-than } (ns, ms) \notin \text{lenlex less-than}$   
**using** *assms eq-omega-sum-less-iff* **by** *metis*  
**moreover have** *total (lenlex less-than)*  
**by** (*simp add: total-lenlex total-less-than*)  
**ultimately show**  $ms = ns$   
**by** (*meson UNIV-I total-on-def*)  
**qed** *auto*

**lemma** *inj-omega-sum*: *inj-on omega-sum {l. length l = n}*  
**unfolding** *inj-on-def* **using** *eq-omega-sum-eq-iff* **by** *fastforce*

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

```

proof (induction n arbitrary:  $\gamma$ )
  case 0
  then show ?case
    by (rule-tac x=[] in exI) auto
next
  case (Suc n)
  then obtain k::nat where k:  $\gamma \in \text{elts } (\omega \uparrow n * k)$ 
    and kmin:  $\bigwedge k'. k' < k \implies \gamma \notin \text{elts } (\omega \uparrow n * k')$ 
    by (metis Ord-ord-of-nat elts-mult- $\omega E$  oexp-succ ord-of-nat.simps(2))
  show ?case
  proof (cases k)
    case (Suc k')
    then obtain  $\delta$  where  $\delta: \gamma = (\omega \uparrow n * k') + \delta$ 
      by (metis lessI mult-succ ord-of-nat.simps(2) k kmin mem-plus-V-E)
    then have  $\delta \text{in}: \delta \in \text{elts } (\omega \uparrow n)$ 
      using Suc k mult-succ by auto
    then obtain ns where ns:  $\delta = \text{omega-sum } ns$  and len:  $\text{length } ns = n$ 
      using Suc.IH by auto
    moreover have  $\text{omega-sum } ns < \omega \uparrow n$ 
      using OrdmemD ns  $\delta \text{in}$  by auto
    ultimately show ?thesis
      by (rule-tac x=k'#ns in exI) (simp add:  $\delta$ )
    qed (use k in auto)
  qed

lemma omega-sum-drop [simp]:  $\text{omega-sum } (\text{dropWhile } (\lambda n. n=0) ns) = \text{omega-sum } ns$ 
  by (induction ns) auto

lemma normal-drop [simp]:  $\text{normal } (\text{dropWhile } (\lambda n. n=0) ns)$ 
  by (induction ns) auto

lemma omega-sum- $\omega\omega$ :
  assumes  $\gamma \in \text{elts } (\omega \uparrow \omega)$ 
  obtains ns where  $\gamma = \text{omega-sum } ns$  normal ns
proof –
  obtain ms where  $\gamma = \text{omega-sum } ms$ 
    using assms Ex-omega-sum by (auto simp: oexp-Limit elts- $\omega$ )
  then show thesis
    by (metis normal-drop omega-sum-drop that)
  qed

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

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

```

by (metis (mono-tags, lifting) Cantor- $\omega$ -def assms omega-sum- $\omega$  someI)+

### 3.2 Larson's set $W(n)$

**definition**  $WW :: \text{nat list set}$   
**where**  $WW \equiv \{l. \text{strict-sorted } l\}$

**fun**  $\text{into-}WW :: \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$  **where**  
 $\text{into-}WW \ k \ [] = []$   
 $|\ \text{into-}WW \ k \ (n\#ns) = (k+n) \# \text{into-}WW \ (\text{Suc } (k+n)) \ ns$

**fun**  $\text{from-}WW :: \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$  **where**  
 $\text{from-}WW \ k \ [] = []$   
 $|\ \text{from-}WW \ k \ (n\#ns) = (n - k) \# \text{from-}WW \ (\text{Suc } n) \ ns$

**lemma**  $\text{from-into-}WW$  [simp]:  $\text{from-}WW \ k \ (\text{into-}WW \ k \ ns) = ns$   
**by** (induction ns arbitrary: k) auto

**lemma**  $\text{inj-into-}WW$ :  $\text{inj} \ (\text{into-}WW \ k)$   
**by** (metis from-into- $WW$  injI)

**lemma**  $\text{into-from-}WW\text{-aux}$ :  
 $\llbracket \text{strict-sorted } ns; \forall n \in \text{list.set } ns. k \leq n \rrbracket \Longrightarrow \text{into-}WW \ k \ (\text{from-}WW \ k \ ns) = ns$   
**by** (induction ns arbitrary: k) (auto simp: Suc-leI)

**lemma**  $\text{into-from-}WW$  [simp]:  $\text{strict-sorted } ns \Longrightarrow \text{into-}WW \ 0 \ (\text{from-}WW \ 0 \ ns) = ns$   
**by** (simp add: into-from- $WW$ -aux)

**lemma**  $\text{into-}WW\text{-imp-ge}$ :  $y \in \text{List.set} \ (\text{into-}WW \ x \ ns) \Longrightarrow x \leq y$   
**by** (induction ns arbitrary: x) fastforce+

**lemma**  $\text{strict-sorted-into-}WW$ :  $\text{strict-sorted} \ (\text{into-}WW \ x \ ns)$   
**by** (induction ns arbitrary: x) (auto simp: dest: into- $WW$ -imp-ge)

**lemma**  $\text{length-into-}WW$ :  $\text{length} \ (\text{into-}WW \ x \ ns) = \text{length } ns$   
**by** (induction ns arbitrary: x) auto

**lemma**  $WW\text{-eq-range-into}$ :  $WW = \text{range} \ (\text{into-}WW \ 0)$

**proof** –

**have**  $\bigwedge ns. \text{strict-sorted } ns \Longrightarrow ns \in \text{range} \ (\text{into-}WW \ 0)$

**by** (metis into-from- $WW$  rangeI)

**then show** ?thesis **by** (auto simp:  $WW$ -def strict-sorted-into- $WW$ )

**qed**

**lemma**  $\text{into-}WW\text{-lenlex-iff}$ :  $(\text{into-}WW \ k \ ms, \text{into-}WW \ k \ ns) \in \text{lenlex less-than} \iff (ms, ns) \in \text{lenlex less-than}$

**proof** (induction ms arbitrary: ns k)

**case** Nil

**then show** *?case*  
 by *simp (metis length-0-conv length-into-WW)*  
**next**  
 case *(Cons m ms)*  
**then show** *?case*  
 by *(induction ns) (auto simp: Cons-lenlex-iff length-into-WW)*  
**qed**

**lemma** *wf-llt [simp]: wf (lenlex less-than) and trans-llt [simp]: trans (lenlex less-than)*  
 by *blast+*

**lemma** *total-llt [simp]: total-on A (lenlex less-than)*  
 by *(meson UNIV-I total-lenlex total-less-than total-on-def)*

**lemma** *omega-sum-1-less:*  
**assumes** *(ms, ns) ∈ lenlex less-than* **shows** *omega-sum (1#ms) < omega-sum (1#ns)*  
**proof** –  
 have *omega-sum (1#ms) < omega-sum (1#ns)* **if** *length ms < length ns*  
 using *omega-sum-less-iff-cases that zero-less-one* **by** *blast*  
**then show** *?thesis*  
 using *assms* **by** *(auto simp: mult-succ simp flip: omega-sum-less-iff)*  
**qed**

**lemma** *ordertype-WW-1: ordertype WW (lenlex less-than) ≤ ordertype UNIV (lenlex less-than)*  
 by *(rule ordertype-mono) auto*

**lemma** *ordertype-WW-2: ordertype UNIV (lenlex less-than) ≤ ω↑ω*  
**proof** *(rule ordertype-inc-le-Ord)*  
**show** *range (λms. omega-sum (1#ms)) ⊆ elts (ω↑ω)*  
 by *(meson Ord-ω Ord-mem-iff-lt Ord-oexp Ord-omega-sum image-subset-iff omega-sum-less-ωω)*  
**qed** *(use omega-sum-1-less in auto)*

**lemma** *ordertype-WW-3: ω↑ω ≤ ordertype WW (lenlex less-than)*  
**proof** –  
**define** *π* **where** *π ≡ into-WW 0 ∘ Cantor-ωω*  
**have** *ωω: ω↑ω = tp (elts (ω↑ω))*  
 by *simp*  
**also have** *... ≤ ordertype WW (lenlex less-than)*  
**proof** *(rule ordertype-inc-le)*  
**fix** *α β*  
**assume** *α: α ∈ elts (ω↑ω) and β: β ∈ elts (ω↑ω) and (α, β) ∈ VWF*  
**then obtain** *\**: *Ord α Ord β α < β*  
 by *(metis Ord-in-Ord Ord-ordertype VWF-iff-Ord-less ωω)*  
**then have** *length (Cantor-ωω α) ≤ length (Cantor-ωω β)*  
**using** *α β* **by** *(simp add: Cantor-ωω normal-Cantor-ωω omega-sum-length-leD)*  
**with** *α β \** **have** *(Cantor-ωω α, Cantor-ωω β) ∈ lenlex less-than*

by (auto simp: Cantor- $\omega$  simp flip: omega-sum-less-iff)  
 then show  $(\pi \alpha, \pi \beta) \in \text{lenlex less-than}$   
 by (simp add:  $\pi$ -def into-WW-lenlex-iff)  
 qed (auto simp:  $\pi$ -def WW-def strict-sorted-into-WW)  
 finally show  $\omega \uparrow \omega \leq \text{ordertype WW (lenlex less-than)}$ .  
 qed

**lemma** ordertype-WW:  $\text{ordertype WW (lenlex less-than)} = \omega \uparrow \omega$   
**and** ordertype-UNIV- $\omega$ :  $\text{ordertype UNIV (lenlex less-than)} = \omega \uparrow \omega$   
**using** ordertype-WW-1 ordertype-WW-2 ordertype-WW-3 **by** auto

**lemma** ordertype- $\omega$ :  
 fixes  $F :: \text{nat} \Rightarrow \text{nat list set}$   
 assumes  $\bigwedge j :: \text{nat}. \text{ordertype (F j) (lenlex less-than)} = \omega \uparrow j$   
 shows  $\text{ordertype } (\bigcup j. F j) \text{ (lenlex less-than)} = \omega \uparrow \omega$   
**proof** (rule antisym)  
 show  $\text{ordertype } (\bigcup (\text{range } F)) \text{ (lenlex less-than)} \leq \omega \uparrow \omega$   
 by (metis ordertype-UNIV- $\omega$  ordertype-mono small top-greatest trans-llt wf-llt)  
 have  $\bigwedge n. \omega \uparrow \text{ord-of-nat } n \leq \text{ordertype } (\bigcup (\text{range } F)) \text{ (lenlex less-than)}$   
 by (metis TC-small Union-upper assms ordertype-mono rangeI trans-llt wf-llt)  
 then show  $\omega \uparrow \omega \leq \text{ordertype } (\bigcup (\text{range } F)) \text{ (lenlex less-than)}$   
 by (auto simp: oexp- $\omega$ -Limit ZFC-in-HOL.SUP-le-iff elts- $\omega$ )  
 qed

**definition** WW-seg  $:: \text{nat} \Rightarrow \text{nat list set}$   
 where  $\text{WW-seg } n \equiv \{l \in \text{WW}. \text{length } l = n\}$

**lemma** WW-seg-subset-WW:  $\text{WW-seg } n \subseteq \text{WW}$   
**by** (auto simp: WW-seg-def)

**lemma** WW-eq-UN-WW-seg:  $\text{WW} = (\bigcup n. \text{WW-seg } n)$   
**by** (auto simp: WW-seg-def)

**lemma** ordertype-list-seg:  $\text{ordertype } \{l. \text{length } l = n\} \text{ (lenlex less-than)} = \omega \uparrow n$   
**proof** –  
 have *bij-betw omega-sum*  $\{l. \text{length } l = n\} \text{ (elts } (\omega \uparrow n))$   
 unfolding WW-seg-def *bij-betw-def*  
 by (auto simp: *inj-omega-sum Ord-mem-iff-lt omega-sum-less dest: Ex-omega-sum*)  
 then show *?thesis*  
 by (force simp: ordertype-eq-iff simp flip: eq-omega-sum-less-iff)  
 qed

**lemma** ordertype-WW-seg:  $\text{ordertype (WW-seg } n) \text{ (lenlex less-than)} = \omega \uparrow n$   
 (is  $\text{ordertype } ?W ?R = \omega \uparrow n$ )

**proof** –  
 have  $\text{ordertype } \{l. \text{length } l = n\} ?R = \text{ordertype } ?W ?R$

```

proof (subst ordertype-eq-ordertype)
  show  $\exists f. \text{bij-betw } f \{l. \text{length } l = n\} ?W \wedge (\forall x \in \{l. \text{length } l = n\}. \forall y \in \{l. \text{length } l = n\}. ((f x, f y) \in \text{lenlex less-than}) = ((x, y) \in \text{lenlex less-than}))$ 
proof (intro exI conjI)
  have inj-on (into-WW 0) {l. length l = n}
  by (metis from-into-WW inj-onI)
  then show bij-betw (into-WW 0) {l. length l = n} ?W
  by (auto simp: bij-betw-def WW-seg-def WW-eq-range-into length-into-WW)
qed (simp add: into-WW-lenlex-iff)
qed auto
then show ?thesis
  using ordertype-list-seg by auto
qed

```

### 3.3 Definitions required for the lemmas

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

```

instantiation list :: (ord)ord
begin

```

```

definition  $xs < ys \equiv xs \neq [] \wedge ys \neq [] \longrightarrow \text{last } xs < \text{hd } ys$  for  $xs \ ys :: 'a \text{ list}$ 

```

```

definition  $xs \leq ys \equiv xs < ys \vee xs = ys$  for  $xs \ ys :: 'a \text{ list}$ 

```

```

instance

```

```

  by standard

```

```

end

```

```

lemma less-Nil [simp]:  $xs < [] \ \ [] < xs$ 
  by (auto simp: less-list-def)

```

```

lemma less-sets-imp-list-less:

```

```

  assumes  $list.set \ xs \ll list.set \ ys$ 

```

```

  shows  $xs < ys$ 

```

```

  by (metis assms last-in-set less-list-def less-sets-def list.set-sel(1))

```

```

lemma less-sets-imp-sorted-list-of-set:

```

```

  assumes  $A \ll B \text{ finite } A \text{ finite } B$ 

```

```

  shows  $list-of \ A < list-of \ B$ 

```

```

  by (simp add: assms less-sets-imp-list-less)

```

```

lemma sorted-list-of-set-imp-less-sets:

```

```

  assumes  $xs < ys \text{ sorted } xs \text{ sorted } ys$ 

```

```

  shows  $list.set \ xs \ll list.set \ ys$ 

```

```

  using assms sorted-hd-le sorted-le-last

```

```

  by (force simp: less-list-def less-sets-def intro: order.trans)

```

```

lemma less-list-iff-less-sets:

```

```

  assumes  $\text{sorted } xs \text{ sorted } ys$ 

```

**shows**  $xs < ys \longleftrightarrow list.set\ xs \ll list.set\ ys$   
**using** *assms sorted-hd-le sorted-le-last*  
**by** (*force simp: less-list-def less-sets-def intro: order.trans*)

**lemma** *strict-sorted-append-iff*:  
 $strict\ sorted\ (xs\ @\ ys) \longleftrightarrow xs < ys \wedge strict\ sorted\ xs \wedge strict\ sorted\ ys$   
**by** (*metis less-list-iff-less-sets less-setsD sorted-wrt-append strict-sorted-imp-less-sets strict-sorted-imp-sorted*)

**lemma** *singleton-less-list-iff*:  $sorted\ xs \implies [n] < xs \longleftrightarrow \{..n\} \cap list.set\ xs = \{\}$   
**apply** (*simp add: less-list-def disjoint-iff*)  
**by** (*metis empty-iff less-le-trans list.set(1) list.set-sel(1) not-le sorted-hd-le*)

**lemma** *less-hd-imp-less*:  $xs < [hd\ ys] \implies xs < ys$   
**by** (*simp add: less-list-def*)

**lemma** *strict-sorted-concat-I*:  
**assumes**  $\bigwedge x. x \in list.set\ xs \implies strict\ sorted\ x$   
 $\bigwedge n. Suc\ n < length\ xs \implies xs!n < xs!Suc\ n$   
 $xs \in lists\ (-\ \{\}\}$   
**shows**  $strict\ sorted\ (concat\ xs)$   
**using** *assms*  
**proof** (*induction xs*)  
**case** (*Cons x xs*)  
**then have**  $x < concat\ xs$   
**apply** (*simp add: less-list-def*)  
**by** (*metis Compl-iff hd-concat insertI1 length-greater-0-conv length-pos-if-in-set list.sel(1) lists.cases nth-Cons-0*)  
**with Cons show** ?*case*  
**by** (*force simp: strict-sorted-append-iff*)  
**qed** *auto*

## 3.4 Nash Williams for lists

### 3.4.1 Thin sets of lists

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

**definition** *thin* :: 'a list set  $\Rightarrow$  bool  
**where**  $thin\ A \equiv \neg (\exists x\ y. x \in A \wedge y \in A \wedge x \neq y \wedge initial\ segment\ x\ y)$

**lemma** *initial-segment-ne*:  
**assumes**  $initial\ segment\ xs\ ys\ xs \neq []$   
**shows**  $ys \neq [] \wedge hd\ ys = hd\ xs$   
**using** *assms* **by** (*auto elim!: initial-segment.cases*)

**lemma** *take-initial-segment*:  
**assumes**  $initial\ segment\ xs\ ys\ k \leq length\ xs$   
**shows**  $take\ k\ xs = take\ k\ ys$



```

by (metis append-eq-conv-conj assms initial-segment.cases min-def take-take)

lemma initial-segment-length-eq:
  assumes initial-segment xs ys length xs = length ys
  shows xs = ys
  using assms initial-segment.cases by fastforce

lemma initial-segment-Nil [simp]: initial-segment [] ys
  by (simp add: initial-segment.simps)

lemma initial-segment-Cons [simp]: initial-segment (x#xs) (y#ys)  $\longleftrightarrow$  x=y  $\wedge$ 
  initial-segment xs ys
  by (metis append-Cons initial-segment.simps list.inject)

lemma init-segment-iff-initial-segment:
  assumes strict-sorted xs strict-sorted ys
  shows init-segment (list.set xs) (list.set ys)  $\longleftrightarrow$  initial-segment xs ys (is ?lhs =
  ?rhs)
proof
  assume ?lhs
  then obtain S' where S': list.set ys = list.set xs  $\cup$  S' list.set xs  $\ll$  S'
    by (auto simp: init-segment-def)
  then have finite S'
    by (metis List.finite-set finite-Un)
  have ys = xs @ list-of S'
    using S'  $\langle$ strict-sorted xs $\rangle$ 
  proof (induction xs)
    case Nil
    with  $\langle$ strict-sorted ys $\rangle$  show ?case
      by auto
    next
    case (Cons a xs)
    with  $\langle$ finite S' $\rangle$  have ys = a # xs @ list-of S'
      by (metis List.finite-set append-Cons assms(2) sorted-list-of-set-Un sorted-list-of-set-set-of)
    then show ?case
      by (auto simp: Cons)
  qed
  then show ?rhs
    using initial-segment.intros by blast
next
  assume ?rhs
  then show ?lhs
  proof cases
    case (1 ys)
    with assms show ?thesis
      by (simp add: init-segment-Un strict-sorted-imp-less-sets)
  qed
qed

```

```

theorem Nash-Williams-WW:
  fixes  $h :: \text{nat list} \Rightarrow \text{nat}$ 
  assumes infinite M and  $h : \{l \in A. \text{List.set } l \subseteq M\} \subseteq \{..<2\}$  and thin A A
   $\subseteq WW$ 
  obtains  $i N$  where  $i < 2$  infinite N  $N \subseteq M$   $h : \{l \in A. \text{List.set } l \subseteq N\} \subseteq \{i\}$ 
proof –
  define AM where  $AM \equiv \{l \in A. \text{List.set } l \subseteq M\}$ 
  have thin-set (list.set ‘A)
    using  $\langle \text{thin } A \rangle \langle A \subseteq WW \rangle$  unfolding thin-def thin-set-def WW-def
    by (auto simp: subset-iff init-segment-iff-initial-segment)
  then have thin-set (list.set ‘AM)
    by (simp add: AM-def image-subset-iff thin-set-def)
  then have Ramsey (list.set ‘AM) 2
    using Nash-Williams-2 by metis
  moreover have  $(h \circ \text{list-of}) \in \text{list.set } ‘AM \rightarrow \{..<2\}$ 
    unfolding AM-def
  proof clarsimp
    fix  $l$ 
    assume  $l \in A$  list.set  $l \subseteq M$ 
    then have strict-sorted  $l$ 
      using WW-def  $\langle A \subseteq WW \rangle$  by blast
    then show  $h (\text{list-of } (\text{list.set } l)) < 2$ 
      using  $h \langle l \in A \rangle \langle \text{list.set } l \subseteq M \rangle$  by auto
  qed
  ultimately obtain  $N i$  where  $N : N \subseteq M$  infinite N  $i < 2$ 
    and list.set ‘AM  $\cap \text{Pow } N \subseteq (h \circ \text{list-of}) - ‘\{i\}$ 
    unfolding Ramsey-eq by (metis  $\langle \text{infinite } M \rangle$ )
  then have N-disjoint:  $(h \circ \text{list-of}) - ‘\{1-i\} \cap (\text{list.set } ‘AM) \cap \text{Pow } N = \{\}$ 
    unfolding subset-vimage-iff less-2-cases-iff by force
  have  $h : \{l \in A. \text{list.set } l \subseteq N\} \subseteq \{i\}$ 
proof clarify
  fix  $l$ 
  assume  $l \in A$  and list.set  $l \subseteq N$ 
  then have  $h l < 2$ 
    using  $h \langle N \subseteq M \rangle$  by force
  with  $\langle i < 2 \rangle$  have  $h l \neq \text{Suc } 0 - i \implies h l = i$ 
    by (auto simp: eval-nat-numeral less-Suc-eq)
  moreover have strict-sorted  $l$ 
    using  $\langle A \subseteq WW \rangle \langle l \in A \rangle$  unfolding WW-def by blast
  moreover have  $h (\text{list-of } (\text{list.set } l)) = 1 - i \longrightarrow \neg (\text{list.set } l \subseteq N)$ 
    using N-disjoint  $\langle N \subseteq M \rangle \langle l \in A \rangle$  by (auto simp: AM-def)
  ultimately show  $h l = i$ 
    using  $N \langle N \subseteq M \rangle \langle l \in A \rangle \langle \text{list.set } l \subseteq N \rangle$ 
    by (auto simp: vimage-def set-eq-iff AM-def WW-def subset-iff)
  qed
  then show thesis
    using that  $\langle i < 2 \rangle N$  by auto
qed

```

### 3.5 Specialised functions on lists

**lemma** *mem-lists-non-Nil*:  $xss \in \text{lists } (- \{\}\}) \longleftrightarrow (\forall x \in \text{list.set } xss. x \neq \{\})$   
**by** *auto*

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

**lemma** *length-acc-lengths* [*simp*]:  $\text{length } (\text{acc-lengths } \text{acc } ls) = \text{length } ls$   
**by** (*induction* *ls* *arbitrary*: *acc*) *auto*

**lemma** *acc-lengths-eq-Nil-iff* [*simp*]:  $\text{acc-lengths } \text{acc } ls = \{\} \longleftrightarrow ls = \{\}$   
**by** (*metis* *length-0-conv* *length-acc-lengths*)

**lemma** *set-acc-lengths*:  
**assumes**  $ls \in \text{lists } (- \{\}\})$  **shows**  $\text{list.set } (\text{acc-lengths } \text{acc } ls) \subseteq \{\text{acc} < ..\}$   
**using** *assms* **by** (*induction* *ls* *rule*: *acc-lengths.induct*) *fastforce*+

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

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

**lemma** *last-acc-lengths* [*simp*]:  
 $ls \neq \{\} \implies \text{last } (\text{acc-lengths } \text{acc } ls) = \text{acc} + \text{sum-list } (\text{map } \text{length } ls)$   
**by** (*induction* *acc* *ls* *rule*: *acc-lengths.induct*) *auto*

**lemma** *nth-acc-lengths* [*simp*]:  
 $\llbracket ls \neq \{\}; k < \text{length } ls \rrbracket \implies \text{acc-lengths } \text{acc } ls ! k = \text{acc} + \text{sum-list } (\text{map } \text{length } (\text{take } (\text{Suc } k) \text{ } ls))$   
**by** (*induction* *acc* *ls* *arbitrary*: *k* *rule*: *acc-lengths.induct*) (*fastforce* *simp*: *less-Suc-eq* *nth-Cons*)+

**lemma** *acc-lengths-plus*:  $\text{acc-lengths } (m+n) \text{ } as = \text{map } ((+)m) (\text{acc-lengths } n \text{ } as)$   
**by** (*induction* *n* *as* *arbitrary*: *m* *rule*: *acc-lengths.induct*) (*auto* *simp*: *add.assoc*)

**lemma** *acc-lengths-shift*: *NO-MATCH*  $0 \text{ } \text{acc} \implies \text{acc-lengths } \text{acc } as = \text{map } ((+) \text{acc}) (\text{acc-lengths } 0 \text{ } as)$   
**by** (*metis* *acc-lengths-plus* *add.comm-neutral*)

**lemma** *length-concat-acc-lengths*:  
 $ls \neq \{\} \implies k + \text{length } (\text{concat } ls) \in \text{list.set } (\text{acc-lengths } k \text{ } ls)$   
**by** (*metis* *acc-lengths-eq-Nil-iff* *last-acc-lengths* *last-in-set* *length-concat*)

**lemma** *strict-sorted-acc-lengths*:  
**assumes**  $ls \in \text{lists } (- \{\}\})$  **shows** *strict-sorted*  $(\text{acc-lengths } \text{acc } ls)$   
**using** *assms*  
**proof** (*induction* *ls* *rule*: *acc-lengths.induct*)  
**case**  $(2 \text{ } \text{acc } l \text{ } ls)$

**then have** *strict-sorted* (*acc-lengths* (*acc + length l*) *ls*)  
**by** *auto*  
**then show** *?case*  
**using** *set-acc-lengths 2.premis* **by** *auto*  
**qed** *auto*

**lemma** *acc-lengths-append*:  
*acc-lengths acc (xs @ ys)*  
= *acc-lengths acc xs @ acc-lengths (acc + sum-list (map length xs)) ys*  
**by** (*induction acc xs rule: acc-lengths.induct*) (*auto simp: add.assoc*)

**lemma** *length-concat-ge*:  
**assumes** *as ∈ lists (- {[]})*  
**shows** *length (concat as) ≥ length as*  
**using** *assms*  
**proof** (*induction as*)  
**case** (*Cons a as*)  
**then have** *length a ≥ Suc 0 ∧ l. l ∈ list.set as ⇒ length l ≥ Suc 0*  
**by** (*auto simp: Suc-leI*)  
**then show** *?case*  
**using** *Cons.IH* **by** *force*  
**qed** *auto*

**fun** *interact* :: '*a* list list ⇒ '*a* list list ⇒ '*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-list (map length xs) + sum-list (map length ys)*  
**by** (*induction rule: interact.induct*) (*auto simp: length-concat*)

**lemma** *length-interact-ge*:  
**assumes** *xs ∈ lists (- {[]}) ys ∈ lists (- {[]})*  
**shows** *length (interact xs ys) ≥ length xs + length ys*  
**by** (*metis add-mono assms length-concat length-concat-ge length-interact*)

**lemma** *set-interact [simp]*:  
**shows** *list.set (interact xs ys) = list.set (concat xs) ∪ list.set (concat ys)*  
**by** (*induction rule: interact.induct*) *auto*

**lemma** *interact-eq-Nil-iff [simp]*:  
**assumes** *xs ∈ lists (- {[]}) ys ∈ lists (- {[]})*  
**shows** *interact xs ys = [] ↔ xs=[] ∧ ys=[]*  
**using** *length-interact-ge [OF assms]* **by** *fastforce*

**lemma** *interact-sing* [*simp*]:  $interact [x] ys = x @ concat ys$   
**by** (*metis* (*no-types*) *concat.simps(2)* *interact.simps* *neq-Nil-conv*)

**lemma** *hd-interact*:  $\llbracket xs \neq []; hd xs \neq [] \rrbracket \implies hd (interact xs ys) = hd (hd xs)$   
**by** (*smt* (*verit*, *best*) *hd-append2* *hd-concat* *interact.elims* *list.sel(1)*)

**lemma** *acc-lengths-concat-injective*:  
**assumes**  $concat as' = concat as$   $acc-lengths n as' = acc-lengths n as$   
**shows**  $as' = as$   
**using** *assms*  
**proof** (*induction as* *arbitrary: n as'*)  
**case** *Nil*  
**then show** *?case*  
**by** (*metis* *acc-lengths-eq-Nil-iff*)  
**next**  
**case** (*Cons a as*)  
**then obtain**  $a' bs$  **where**  $as' = a' \# bs$   
**by** (*metis* *Suc-length-conv* *length-acc-lengths*)  
**with** *Cons* **show** *?case*  
**by** *simp*  
**qed**

**lemma** *acc-lengths-interact-injective*:  
**assumes**  $interact as' bs' = interact as bs$   $acc-lengths a as' = acc-lengths a as$   
 $acc-lengths b bs' = acc-lengths b bs$   
**shows**  $as' = as \wedge bs' = bs$   
**using** *assms*  
**proof** (*induction as bs* *arbitrary: a b as' bs'* *rule: interact.induct*)  
**case** (*1 cs*) **then show** *?case*  
**by** (*metis* *acc-lengths-concat-injective* *acc-lengths-eq-Nil-iff* *interact.simps(1)*)  
**next**  
**case** (*2 c cs*)  
**then show** *?case*  
**by** (*metis* *acc-lengths-concat-injective* *acc-lengths-eq-Nil-iff* *interact.simps(2)*)  
*list.exhaust*)  
**next**  
**case** (*3 x xs y ys*)  
**then obtain**  $a' us b' vs$  **where**  $as' = a' \# us$   $bs' = b' \# vs$   
**by** (*metis* *length-Suc-conv* *length-acc-lengths*)  
**with** *3* **show** *?case*  
**by** *auto*  
**qed**

**lemma** *strict-sorted-interact-I*:  
**assumes**  $length ys \leq length xs$   $length xs \leq Suc (length ys)$   
 $\bigwedge x. x \in list.set xs \implies strict-sorted x$   
 $\bigwedge y. y \in list.set ys \implies strict-sorted y$   
 $\bigwedge n. n < length ys \implies xs!n < ys!n$

```

   $\wedge n. \text{Suc } n < \text{length } xs \implies ys!n < xs!\text{Suc } n$ 
assumes  $xs \in \text{lists } (- \{\}\}$   $ys \in \text{lists } (- \{\}\}$ 
shows strict-sorted (interact  $xs$   $ys$ )
using assms
proof (induction rule: interact.induct)
case ( $\exists x$   $xs$   $y$   $ys$ )
then have  $x < y$ 
  by force
moreover have strict-sorted (interact  $xs$   $ys$ )
  using  $\exists$  by simp (metis Suc-less-eq nth-Cons-Suc)
moreover have  $y < \text{interact } xs$   $ys$ 
  using  $\exists$  apply (simp add: less-list-def)
  by (metis hd-interact le-zero-eq length-greater-0-conv list.sel(1) list.set-sel(1)
list.size(3) lists.simps mem-lists-non-Nil nth-Cons-0)
  ultimately show ?case
  using  $\exists$  by (simp add: strict-sorted-append-iff less-list-def)
qed auto

```

## 3.6 Forms and interactions

### 3.6.1 Forms

```

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

```

```

inductive Form :: [nat, nat list set]  $\Rightarrow$  bool
where Form  $0$   $\{xs, ys\}$  if  $\text{length } xs = \text{length } ys$   $xs \neq ys$ 
  | Form  $(2*k-1)$   $\{xs, ys\}$  if Form-Body  $k$   $k$   $xs$   $ys$   $zs$   $k > 0$ 
  | Form  $(2*k)$   $\{xs, ys\}$  if Form-Body  $(\text{Suc } k)$   $k$   $xs$   $ys$   $zs$   $k > 0$ 

```

```

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

```

```

lemma Form-elim-upair:

```

```

assumes Form  $l$   $U$ 

```

```

obtains  $xs$   $ys$  where  $xs \neq ys$   $U = \{xs, ys\}$   $\text{length } xs \leq \text{length } ys$ 

```

```

using assms

```

```

by (smt (verit, best) Form.simps Form-Body.cases less-or-eq-imp-le nat-neq-iff)

```

```

lemma assumes Form-Body  $ka$   $kb$   $xs$   $ys$   $zs$ 

```

```

shows Form-Body-WW:  $zs \in WW$ 

```

```

and Form-Body-nonempty:  $\text{length } zs > 0$ 

```

**and** *Form-Body-length*:  $\text{length } xs < \text{length } ys$   
**using** *Form-Body.cases* [*OF assms*] **by** (*fastforce simp: WW-def*)<sup>+</sup>

**lemma** *form-cases*:

**fixes**  $l::\text{nat}$

**obtains** (*zero*)  $l = 0 \mid$  (*nz*)  $ka \ kb$  **where**  $l = ka+kb - 1 \ 0 < kb \ kb \leq ka \ ka \leq \text{Suc } kb$

**proof** –

**have**  $l = 0 \vee (\exists ka \ kb. l = ka+kb - 1 \wedge 0 < kb \wedge kb \leq ka \wedge ka \leq \text{Suc } kb)$

**by** *presburger*

**then show** *thesis*

**using** *nz zero* **by** *blast*

**qed**

### 3.6.2 Interactions

**lemma** *interact*:

**assumes** *Form l U l>0*

**obtains**  $ka \ kb \ xs \ ys \ zs$  **where**  $l = ka+kb - 1 \ U = \{xs,ys\} \ \text{Form-Body } ka \ kb \ xs \ ys \ 0 < kb \ kb \leq ka \ ka \leq \text{Suc } kb$

**using** *assms*

**unfolding** *Form.simps*

**by** (*smt (verit, best) add-Suc diff-Suc-1 lessI mult-2 nat-less-le order-refl*)

**definition** *inter-scheme* ::  $\text{nat} \Rightarrow \text{nat list set} \Rightarrow \text{nat list}$

**where** *inter-scheme l U*  $\equiv$

*SOME*  $zs. \exists k \ xs \ ys. U = \{xs,ys\} \wedge$

$(l = 2*k-1 \wedge \text{Form-Body } k \ k \ xs \ ys \ zs \vee l = 2*k \wedge \text{Form-Body } (\text{Suc } k)$

$k \ xs \ ys \ zs)$

**lemma** *inter-scheme*:

**assumes** *Form l U l>0*

**obtains**  $ka \ kb \ xs \ ys$  **where**  $l = ka+kb - 1 \ U = \{xs,ys\} \ \text{Form-Body } ka \ kb \ xs \ ys$   
(*inter-scheme l U*)  $0 < kb \ kb \leq ka \ ka \leq \text{Suc } kb$

**using** *interact* [*OF (Form l U)*]

**proof** *cases*

**case** (*2 ka kb xs ys zs*)

**then have**  $\S: \bigwedge ka \ kb \ zs. \neg \text{Form-Body } ka \ kb \ ys \ xs \ zs$

**using** *Form-Body-length less-asymp'* **by** *blast*

**have** *Form-Body ka kb xs ys (inter-scheme l U)*

**proof** (*cases ka = kb*)

**case** *True*

**with** *2* **have**  $l: \forall k. l \neq k * 2$

**by** *presburger*

**have** [*simp*]:  $\bigwedge k. kb + kb - \text{Suc } 0 = k * 2 - \text{Suc } 0 \longleftrightarrow k=kb$

**by** *auto*

**show** *?thesis*

```

    unfolding inter-scheme-def using 2 l True
    by (auto simp: § ⟨l > 0⟩ Set.doubleton-eq-iff conj-disj-distribR ex-disj-distrib
algebra-simps some-eq-ex)
  next
    case False
    with 2 have l: ∀k. l ≠ k * 2 - Suc 0 and [simp]: ka = Suc kb
      by presburger+
    have [simp]: ∧k. kb + kb = k * 2 ⟷ k=kb
      by auto
    show ?thesis
      unfolding inter-scheme-def using 2 l False
      by (auto simp: § ⟨l > 0⟩ Set.doubleton-eq-iff conj-disj-distribR ex-disj-distrib
algebra-simps some-eq-ex)
  qed
  then show ?thesis
    by (simp add: 2 that)
qed (use ⟨l > 0⟩ in auto)

```

```

lemma inter-scheme-strict-sorted:
  assumes Form l U l>0
  shows strict-sorted (inter-scheme l U)
  using Form-Body.simps assms inter-scheme by fastforce

```

```

lemma inter-scheme-simple:
  assumes Form l U l>0
  shows inter-scheme l U ∈ WW ∧ length (inter-scheme l U) > 0
  using inter-scheme [OF assms] by (meson Form-Body-WW Form-Body-nonempty)

```

### 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 -

```

```

  obtain ka kb xs ys

```

```

  where l: l = ka+kb - 1 and U: U = {xs,ys}

```

```

  and FB: Form-Body ka kb xs ys (inter-scheme l U)

```

```

  and kb: 0 < kb kb ≤ ka ka ≤ Suc kb

```

```

  using assms inter-scheme by blast

```

```

  then obtain a as b bs c d

```

```

  where xs: xs = concat (a#as) and ys: ys = concat (b#bs)

```

```

  and len: length (a#as) = ka length (b#bs) = kb

```

```

  and c: c = acc-lengths 0 (a#as)

```

```

  and d: d = acc-lengths 0 (b#bs)

```

```

  and Ueq: inter-scheme l U = concat [c, a, d, b] @ interact as bs

```

```

  by (auto simp: Form-Body.simps)

```

```

  obtain ka' kb' xs' ys'

```

```

  where l': l = ka'+kb' - 1 and U': U' = {xs',ys'}

```



**and** *FB'*: *Form-Body*  $ka' kb' xs' ys'$  (*inter-scheme*  $l U'$ )  
**and**  $kb'$ :  $0 < kb' kb' \leq ka' ka' \leq \text{Suc } kb'$   
**using** *assms inter-scheme* **by** *blast*  
**then obtain**  $a' as' b' bs' c' d'$   
**where**  $xs'$ :  $xs' = \text{concat } (a' \# as')$  **and**  $ys'$ :  $ys' = \text{concat } (b' \# bs')$   
**and**  $len'$ :  $\text{length } (a' \# as') = ka' \text{ length } (b' \# bs') = kb'$   
**and**  $c'$ :  $c' = \text{acc-lengths } 0 (a' \# as')$   
**and**  $d'$ :  $d' = \text{acc-lengths } 0 (b' \# bs')$   
**and**  $Ueq'$ : *inter-scheme*  $l U' = \text{concat } [c', a', d', b'] @ \text{interact } as' bs'$   
**using** *Form-Body.simps* **by** *auto*  
**have** [*simp*]:  $ka' = ka \wedge kb' = kb$   
**using**  $\langle l > 0 \rangle l l' kb kb' le\text{-SucE } le\text{-antisym } mult\text{-2}$  **by** *linarith*  
**have** [*simp*]:  $\text{length } c = \text{length } c' \text{ length } d = \text{length } d'$   
**using**  $c c' d d' len' len$  **by** *auto*  
**have** *c-off*:  $c' = c a' @ d' @ b' @ \text{interact } as' bs' = a @ d @ b @ \text{interact } as bs$   
**using** *eq* **by** (*auto simp: Ueq Ueq'*)  
**then have** *len-a*:  $\text{length } a' = \text{length } a$   
**by** (*metis acc-lengths.simps(2) add.left-neutral c c' nth-Cons-0*)  
**with** *c-off* **have**  $\S$ :  $a' = a d' = d b' @ \text{interact } as' bs' = b @ \text{interact } as bs$   
**by** *auto*  
**then have**  $\text{length } (\text{interact } as' bs') = \text{length } (\text{interact } as bs)$   
**by** (*metis acc-lengths.simps(2) add-left-cancel append-eq-append-conv d d' list.inject*)  
**with**  $\S$  **have**  $b' = b \text{ interact } as' bs' = \text{interact } as bs$   
**by** *auto*  
**moreover have**  $\text{acc-lengths } 0 as' = \text{acc-lengths } 0 as$   
**using**  $\langle a' = a \rangle \langle c' = c \rangle$  **by** (*simp add: c' c acc-lengths-shift*)  
**moreover have**  $\text{acc-lengths } 0 bs' = \text{acc-lengths } 0 bs$   
**using**  $\langle b' = b \rangle \langle d' = d \rangle$  **by** (*simp add: d' d acc-lengths-shift*)  
**ultimately have**  $as' = as \wedge bs' = bs$   
**using** *acc-lengths-interact-injective* **by** *blast*  
**then show** *?thesis*  
**by** (*simp add: \langle a' = a \rangle U U' \langle b' = b \rangle xs xs' ys ys'*)  
**qed**

**lemma** *strict-sorted-interact-imp-concat*:

*strict-sorted (interact as bs)  $\implies$  strict-sorted (concat as)  $\wedge$  strict-sorted (concat bs)*

**proof** (*induction as bs rule: interact.induct*)

**case**  $(\exists x xs y ys)$

**have**  $x < \text{concat } xs$

**using** *3.prem*s

**by** (*smt (verit, del-insts) Un-iff hd-in-set interact.simps(3) last-in-set less-list-def set-append set-interact sorted-wrt-append*)

**moreover have**  $y < \text{concat } ys$

**using** *3 sorted-wrt-append strict-sorted-append-iff* **by** *fastforce*

**ultimately show** *?case*

**using** *3* **by** (*auto simp add: strict-sorted-append-iff*)

**qed** *auto*

**lemma** *strict-sorted-interact-hd*:  
 $\llbracket \text{strict-sorted } (\text{interact } cs \ ds); \ cs \neq []; \ ds \neq []; \ hd \ cs \neq []; \ hd \ ds \neq [] \rrbracket$   
 $\implies \text{hd } (\text{hd } cs) < \text{hd } (\text{hd } ds)$   
**by** (*metis append-is-Nil-conv hd-append2 hd-in-set interact.simps(3) list.exhaust-sel sorted-wrt-append*)

the lengths of the two lists can differ by one

**proposition** *interaction-scheme-unique-aux*:  
**assumes**  $\text{concat } as = \text{concat } as'$  **and**  $ys'$ :  $\text{concat } bs = \text{concat } bs'$   
**and**  $as \in \text{lists } (- \{\})$   $bs \in \text{lists } (- \{\})$   
**and** *strict-sorted* (*interact*  $as \ bs$ )  
**and**  $\text{length } bs \leq \text{length } as$   $\text{length } as \leq \text{Suc } (\text{length } bs)$   
**and**  $as' \in \text{lists } (- \{\})$   $bs' \in \text{lists } (- \{\})$   
**and** *strict-sorted* (*interact*  $as' \ bs'$ )  
**and**  $\text{length } bs' \leq \text{length } as'$   $\text{length } as' \leq \text{Suc } (\text{length } bs')$   
**and**  $\text{length } as = \text{length } as'$   $\text{length } bs = \text{length } bs'$   
**shows**  $as = as' \wedge bs = bs'$   
**using** *assms*  
**proof** (*induction length as arbitrary: as bs as' bs'*)  
**case 0 then show** *?case*  
**by** *auto*  
**next**  
**case SUC:** (*Suc k*)  
**show** *?case*  
**proof** (*cases k*)  
**case 0**  
**with SUC obtain**  $a \ a'$  **where**  $aa'$ :  $as = [a]$   $as' = [a']$   
**by** (*metis Suc-length-conv length-0-conv*)  
**show** *?thesis*  
**proof**  
**show**  $as = as'$   
**using**  $aa'$   $\langle \text{concat } as = \text{concat } as' \rangle$  **by** *force*  
**with SUC 0 show**  $bs = bs'$   
**by** (*metis Suc-leI append-Nil2 concat.simps impossible-Cons le-antisym length-greater-0-conv list.exhaust*)  
**qed**  
**next**  
**case** (*Suc k'*)  
**then obtain**  $a \ cs \ b \ ds$  **where**  $eq$ :  $as = a \# \ cs$   $bs = b \# \ ds$   
**using** *SUC*  
**by** (*metis le0 list.exhaust list.size(3) not-less-eq-eq*)  
**have**  $\text{length } as' \neq 0$   
**using** *SUC* **by** *force*  
**then obtain**  $a' \ cs' \ b' \ ds'$  **where**  $eq'$ :  $as' = a' \# \ cs'$   $bs' = b' \# \ ds'$   
**by** (*metis*  $\langle \text{length } bs = \text{length } bs' \rangle$   $eq(2)$  *length-0-conv list.exhaust*)  
**obtain**  $k$ :  $k = \text{length } cs$   $k \leq \text{Suc } (\text{length } ds)$   
**using**  $eq$  *SUC* **by** *auto*

```

case (Suc k')
obtain [simp]: b ≠ [] b' ≠ [] a ≠ [] a' ≠ []
  using SUC by (simp add: eq eq')
then have hd b' = hd b
  using SUC by (metis concat.simps(2) eq'(2) eq(2) hd-append2)
have ss-ab: strict-sorted (concat as) strict-sorted (concat bs)
  using strict-sorted-interact-imp-concat SUC.prem(5) by blast+
have sw-ab: strict-sorted (a @ b @ interact cs ds)
  by (metis SUC.prem(5) eq interact.simps(3))
then obtain a < b strict-sorted a strict-sorted b
  by (metis append-assoc strict-sorted-append-iff)
have b-cs: strict-sorted (concat (b # cs))
  by (metis append.simps(1) concat.simps(2) interact.simps(3) strict-sorted-interact-imp-concat
sw-ab)
then have b < concat cs
  using strict-sorted-append-iff by auto
have strict-sorted (a @ concat cs)
  using eq(1) ss-ab(1) by force
have list.set a = list.set (concat as) ∩ {..

```

```

    if  $x < \text{hd } b'$  and  $l \in \text{list.set } cs'$  and  $x \in \text{list.set } l$  for  $x l$ 
    using  $b\text{-}cs'$  sorted-hd-le strict-sorted-imp-sorted that by fastforce
    then show ?thesis
    using  $\langle b' \neq [] \rangle$  sw-ab' by (force simp: strict-sorted-append-iff sorted-wrt-append
eq^)
qed
ultimately have  $a=a'$ 
  by (simp add: SUC.premis(1)  $\langle \text{hd } b' = \text{hd } b \rangle$   $\langle \text{strict-sorted } a' \rangle$   $\langle \text{strict-sorted } a \rangle$ 
  strict-sorted-equal)
moreover
have ccat-cs-cs':  $\text{concat } cs = \text{concat } cs'$ 
  using SUC.premis(1)  $\langle a = a' \rangle$  eq'(1) eq(1) by fastforce
have  $b=b'$ 
proof (cases  $ds = [] \vee ds' = []$ )
  case True
  then show ?thesis
    using SUC eq'(2) eq(2) by fastforce
  next
  case False
  then have  $ds \neq []$   $ds' \neq []$  sorted (concat ds) sorted (concat ds')
  using eq(2) ss-ab(2) eq'(2) ss-ab'(2) strict-sorted-append-iff strict-sorted-imp-sorted
  by auto
  have strict-sorted b strict-sorted b'
    using b-cs b-cs' sorted-wrt-append by auto
  moreover
  have  $cs \neq []$ 
    using k local.Suc by auto
  then obtain  $\text{hd } cs \neq []$   $\text{hd } ds \neq []$ 
    using SUC.premis(3) SUC.premis(4) eq list.set-sel(1)
    by (simp add:  $\langle ds \neq [] \rangle$  mem-lists-non-Nil)
  then have  $\text{concat } cs \neq []$ 
    using  $\langle cs \neq [] \rangle$  hd-in-set by auto
  have  $\text{hd } (\text{concat } cs) < \text{hd } (\text{concat } ds)$ 
    using strict-sorted-interact-hd
  by (metis  $\langle cs \neq [] \rangle$   $\langle ds \neq [] \rangle$   $\langle \text{hd } cs \neq [] \rangle$   $\langle \text{hd } ds \neq [] \rangle$  hd-concat sorted-wrt-append
sw-ab)

  have  $\text{list.set } b = \text{list.set } (\text{concat } bs) \cap \{.. < \text{hd } (\text{concat } cs)\}$ 
  proof -
    have 1:  $x \in \text{list.set } b$ 
      if  $x < \text{hd } (\text{concat } cs)$  and  $l \in \text{list.set } ds$  and  $x \in \text{list.set } l$  for  $x l$ 
      using  $\langle \text{hd } (\text{concat } cs) < \text{hd } (\text{concat } ds) \rangle$   $\langle \text{sorted } (\text{concat } ds) \rangle$  sorted-hd-le
    that by fastforce
    have 2:  $l < \text{hd } (\text{concat } cs)$  if  $l \in \text{list.set } b$  for  $l$ 
    by (metis  $\langle \text{concat } cs \neq [] \rangle$  b-cs concat.simps(2) list.set-sel(1) sorted-wrt-append
    that)
    show ?thesis
      using 1 2 by (auto simp: strict-sorted-append-iff sorted-wrt-append eq)
  qed

```

```

moreover
have  $cs' \neq []$ 
  using  $k \text{ Suc } \langle \text{concat } cs \neq [] \rangle \text{ ccat-cs-cs' by auto}$ 
then obtain  $hd \ cs' \neq [] \ hd \ ds' \neq []$ 
  using  $SUC.prem5(8,9) \langle ds' \neq [] \rangle \text{ eq'(1) eq'(2) list.set-sel(1) by auto}$ 
then have  $\text{concat } cs' \neq []$ 
  using  $\langle cs' \neq [] \rangle \text{ hd-in-set by auto}$ 
have  $hd \ (\text{concat } cs') < hd \ (\text{concat } ds')$ 
  using  $\text{strict-sorted-interact-hd}$ 
  by  $(metis \langle cs' \neq [] \rangle \langle ds' \neq [] \rangle \langle hd \ cs' \neq [] \rangle \langle hd \ ds' \neq [] \rangle \text{hd-concat}$ 
 $\text{sorted-wrt-append sw-ab'})$ 
have  $\text{list.set } b' = \text{list.set } (\text{concat } bs') \cap \{..< hd \ (\text{concat } cs')\}$ 
proof –
  have  $1: x \in \text{list.set } b'$ 
    if  $x < hd \ (\text{concat } cs')$  and  $l \in \text{list.set } ds'$  and  $x \in \text{list.set } l$  for  $x \ l$ 
    using  $\langle hd \ (\text{concat } cs') < hd \ (\text{concat } ds') \rangle \langle \text{sorted } (\text{concat } ds') \rangle \text{sorted-hd-le}$ 
that by  $\text{fastforce}$ 
    have  $2: l < hd \ (\text{concat } cs')$  if  $l \in \text{list.set } b'$  for  $l$ 
    by  $(metis \langle \text{concat } cs' \neq [] \rangle \text{b-cs' list.set-sel(1) sorted-wrt-append that})$ 
    show  $?thesis$ 
    using  $1 \ 2 \text{ by (auto simp: strict-sorted-append-iff sorted-wrt-append eq')}$ 
qed
ultimately show  $b = b'$ 
  by  $(\text{simp add: } SUC.prem5(2) \text{ ccat-cs-cs' strict-sorted-equal})$ 
qed
moreover
have  $cs = cs' \wedge ds = ds'$ 
proof  $(\text{rule } SUC.hyps)$ 
  show  $k = \text{length } cs$ 
    using  $\text{eq } SUC.hyps(2) \text{ by auto}[1]$ 
  show  $\text{concat } ds = \text{concat } ds'$ 
    using  $SUC.prem5(2) \langle b = b' \rangle \text{eq'(2) eq(2) by auto}$ 
  show  $\text{strict-sorted } (\text{interact } cs \ ds)$ 
    using  $\text{eq } SUC.prem5(5) \text{strict-sorted-append-iff by auto}$ 
  show  $\text{length } ds \leq \text{length } cs \ \text{length } cs \leq \text{Suc } (\text{length } ds)$ 
    using  $\text{eq } SUC \ k \text{ by auto}$ 
  show  $\text{strict-sorted } (\text{interact } cs' \ ds')$ 
    using  $\text{eq' } SUC.prem5(10) \text{strict-sorted-append-iff by auto}$ 
  show  $\text{length } cs = \text{length } cs'$ 
    using  $SUC \ \text{eq'(1) } k(1) \text{ by force}$ 
qed  $(\text{use ccat-cs-cs' eq eq' } SUC.prem5 \text{ in auto})$ 
ultimately show  $?thesis$ 
  by  $(\text{simp add: } \langle a = a' \rangle \langle b = b' \rangle \text{eq eq'})$ 
qed
qed

```

**proposition** *Form-Body-unique:*

**assumes** *Form-Body*  $ka \ kb \ xs \ ys \ zs$  *Form-Body*  $ka \ kb \ xs \ ys \ zs'$  **and**  $kb \leq ka \ ka \leq$

```

Suc kb
shows  $zs' = zs$ 
proof –
  obtain  $a\ as\ b\ bs\ c\ d$ 
    where  $xs: xs = \text{concat } (a\#\ as)$  and  $ys: ys = \text{concat } (b\#\ bs)$ 
    and  $ne: a\#\ as \in \text{lists } (-\ \{\ \})$   $b\#\ bs \in \text{lists } (-\ \{\ \})$ 
    and  $len: \text{length } (a\#\ as) = ka$   $\text{length } (b\#\ bs) = kb$ 
    and  $c: c = \text{acc-lengths } 0\ (a\#\ as)$ 
    and  $d: d = \text{acc-lengths } 0\ (b\#\ bs)$ 
    and  $Ueq: zs = \text{concat } [c, a, d, b] \text{ @ interact } as\ bs$ 
    and  $ss\text{-}zs: \text{strict-sorted } zs$ 
  using Form-Body.cases [OF assms(1)] by (metis (no-types))
  obtain  $a'\ as'\ b'\ bs'\ c'\ d'$ 
    where  $xs': xs = \text{concat } (a'\#\ as')$  and  $ys': ys = \text{concat } (b'\#\ bs')$ 
    and  $ne': a'\#\ as' \in \text{lists } (-\ \{\ \})$   $b'\#\ bs' \in \text{lists } (-\ \{\ \})$ 
    and  $len': \text{length } (a'\#\ as') = ka$   $\text{length } (b'\#\ bs') = kb$ 
    and  $c': c' = \text{acc-lengths } 0\ (a'\#\ as')$ 
    and  $d': d' = \text{acc-lengths } 0\ (b'\#\ bs')$ 
    and  $Ueq': zs' = \text{concat } [c', a', d', b'] \text{ @ interact } as'\ bs'$ 
    and  $ss\text{-}zs': \text{strict-sorted } zs'$ 
  using Form-Body.cases [OF assms(2)] by (metis (no-types))
  have [simp]:  $\text{length } c = \text{length } c'$   $\text{length } d = \text{length } d'$ 
    using  $c\ c'\ d\ d'\ len'\ len$  by auto
  note acc-lengths.simps [simp del]
  have  $a < b$ 
    using  $ss\text{-}zs$  by (auto simp: Ueq strict-sorted-append-iff less-list-def c d)
  have  $a' < b'$ 
    using  $ss\text{-}zs'$  by (auto simp: Ueq' strict-sorted-append-iff less-list-def c' d')
  have  $a\#\ as = a'\#\ as' \wedge b\#\ bs = b'\#\ bs'$ 
  proof (rule interaction-scheme-unique-aux)
    show strict-sorted (interact ( $a\ \#\ as$ ) ( $b\ \#\ bs$ ))
      using  $ss\text{-}zs\ \langle a < b \rangle$  by (auto simp: Ueq strict-sorted-append-iff less-list-def d)
    show strict-sorted (interact ( $a'\ \#\ as'$ ) ( $b'\ \#\ bs'$ ))
      using  $ss\text{-}zs'\ \langle a' < b' \rangle$  by (auto simp: Ueq' strict-sorted-append-iff less-list-def
d')
    show  $\text{length } (b\ \#\ bs) \leq \text{length } (a\ \#\ as)$   $\text{length } (b'\ \#\ bs') \leq \text{length } (a'\ \#\ as')$ 
      using  $\langle kb \leq ka \rangle\ len\ len'$  by auto
    show  $\text{length } (a\ \#\ as) \leq \text{Suc } (\text{length } (b\ \#\ bs))$ 
      using  $\langle ka \leq \text{Suc } kb \rangle\ len$  by linarith
    then show  $\text{length } (a'\ \#\ as') \leq \text{Suc } (\text{length } (b'\ \#\ bs'))$ 
      using  $len\ len'$  by fastforce
  qed (use len len' xs xs' ys ys' ne ne' in fastforce)
  then show ?thesis
    using  $Ueq\ Ueq'\ c\ c'\ d\ d'$  by blast
qed

```

**lemma** *Form-Body-imp-inter-scheme*:

**assumes** *FB: Form-Body*  $ka\ kb\ xs\ ys\ zs$  **and**  $0 < kb\ kb \leq ka\ ka \leq \text{Suc } kb$

```

shows  $zs = \text{inter-scheme } ((ka+kb) - \text{Suc } 0) \{xs,ys\}$ 
proof -
  have  $\text{length } xs < \text{length } ys$ 
    by (meson Form-Body-length assms(1))
  have [simp]:  $a + a = b + b \longleftrightarrow a=b \quad a + a - \text{Suc } 0 = b + b - \text{Suc } 0 \longleftrightarrow$ 
 $a=b$  for  $a b::\text{nat}$ 
    by auto
  show ?thesis
  proof (cases ka = kb)
    case True
      show ?thesis
        unfolding inter-scheme-def
        apply (rule some-equality [symmetric],metis One-nat-def True FB mult-2)
        using assms  $\langle \text{length } xs < \text{length } ys \rangle$ 
        by (auto simp: True mult-2 Set.doubleton-eq-iff Form-Body-unique dest:
Form-Body-length, presburger)
      next
        case False
        then have  $eq: ka = \text{Suc } kb$ 
          using assms by linarith
        show ?thesis
          unfolding inter-scheme-def
          apply (rule some-equality [symmetric], use assms False mult-2 one-is-add eq)
in fastforce)
    using assms  $\langle \text{length } xs < \text{length } ys \rangle$ 
    by (auto simp: eq mult-2 Set.doubleton-eq-iff Form-Body-unique dest: Form-Body-length,
presburger)
  qed
qed

```

### 3.7 For Lemma 3.8 AND PROBABLY 3.7

**definition**  $\text{grab} :: \text{nat set} \Rightarrow \text{nat} \Rightarrow \text{nat set} \times \text{nat set}$   
**where**  $\text{grab } N n \equiv (N \cap \text{enumerate } N \text{ ' } \{..<n\}, N \cap \{\text{enumerate } N n..\})$

**lemma**  $\text{grab-0}$  [*simp*]:  $\text{grab } N 0 = (\{\}, N)$   
**by** (*fastforce simp: grab-def enumerate-0 Least-le*)

**lemma**  $\text{less-sets-grab}$ :  
 $\text{infinite } N \Longrightarrow \text{fst } (\text{grab } N n) \ll \text{snd } (\text{grab } N n)$   
**by** (*auto simp: grab-def less-sets-def intro: enumerate-mono less-le-trans*)

**lemma**  $\text{finite-grab}$  [*iff*]:  $\text{finite } (\text{fst } (\text{grab } N n))$   
**by** (*simp add: grab-def*)

**lemma**  $\text{card-grab}$  [*simp*]:  
**assumes**  $\text{infinite } N$  **shows**  $\text{card } (\text{fst } (\text{grab } N n)) = n$   
**proof** -  
**have**  $N \cap \text{enumerate } N \text{ ' } \{..<n\} = \text{enumerate } N \text{ ' } \{..<n\}$

**using** *assms* **by** (*auto simp: enumerate-in-set*)  
**with** *assms* **show** *?thesis*  
**by** (*simp add: card-image grab-def strict-mono-enum strict-mono-imp-inj-on*)  
**qed**

**lemma** *fst-grab-subset*:  $\text{fst } (\text{grab } N \ n) \subseteq N$   
**using** *grab-def range-enum* **by** *fastforce*

**lemma** *snd-grab-subset*:  $\text{snd } (\text{grab } N \ n) \subseteq N$   
**by** (*auto simp: grab-def*)

**lemma** *grab-Un-eq*:  
**assumes** *infinite N* **shows**  $\text{fst } (\text{grab } N \ n) \cup \text{snd } (\text{grab } N \ n) = N$   
**proof**  
**show**  $N \subseteq \text{fst } (\text{grab } N \ n) \cup \text{snd } (\text{grab } N \ n)$   
**unfolding** *grab-def*  
**using** *assms enumerate-Ex le-less-linear strict-mono-enum strict-mono-less* **by**  
*fastforce*  
**qed** (*simp add: grab-def*)

**lemma** *finite-grab-iff* [*simp*]:  $\text{finite } (\text{snd } (\text{grab } N \ n)) \longleftrightarrow \text{finite } N$   
**by** (*metis finite-grab grab-Un-eq infinite-Un infinite-super snd-grab-subset*)

**lemma** *grab-eqD*:  
 $\llbracket \text{grab } N \ n = (A, M); \text{infinite } N \rrbracket$   
 $\implies A \ll M \wedge \text{finite } A \wedge \text{card } A = n \wedge \text{infinite } M \wedge A \subseteq N \wedge M \subseteq N$   
**using** *card-grab grab-def less-sets-grab finite-grab-iff* **by** *auto*

**lemma** *less-sets-fst-grab*:  $A \ll N \implies A \ll \text{fst } (\text{grab } N \ n)$   
**by** (*simp add: fst-grab-subset less-sets-weaken2*)

Possibly redundant, given *grab*

**definition** *nxt* **where**  $\text{nxt} \equiv \lambda N. \lambda n::\text{nat}. N \cap \{n<..\}$

**lemma** *infinite-nxtN*:  $\text{infinite } N \implies \text{infinite } (\text{nxt } N \ n)$   
**by** (*simp add: infinite-nat-greaterThan nxt-def*)

**lemma** *nxt-subset*:  $\text{nxt } N \ n \subseteq N$   
**unfolding** *nxt-def* **by** *blast*

**lemma** *nxt-subset-greaterThan*:  $m \leq n \implies \text{nxt } N \ n \subseteq \{m<..\}$   
**by** (*auto simp: nxt-def*)

**lemma** *nxt-subset-atLeast*:  $m \leq n \implies \text{nxt } N \ n \subseteq \{m..\}$   
**by** (*auto simp: nxt-def*)

**lemma** *enum-nxt-ge*:  $\text{infinite } N \implies a \leq \text{enum } (\text{nxt } N \ a) \ n$   
**by** (*simp add: atLeast-le-enum infinite-nxtN nxt-subset-atLeast*)



**lemma** *inj-enum-nxt*:  $\text{infinite } N \implies \text{inj-on } (\text{enum } (\text{nxt } N \ a)) \ A$   
**by** (*simp add: infinite-nxtN strict-mono-enum strict-mono-imp-inj-on*)

### 3.8 Larson's Lemma 3.11

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

**lemma** *lemma-3-11*:

**assumes**  $l > 0$

**shows** *thin* (*inter-scheme*  $l \ \{U. \text{Form } l \ U\}$ )

**using** *form-cases* [of  $l$ ]

**proof** *cases*

**case** *zero*

**then show** *?thesis*

**using** *assms* **by** *auto*

**next**

**case** (*nz ka kb*)

**note** *acc-lengths.simps* [*simp del*]

**show** *?thesis*

**unfolding** *thin-def*

**proof** *clarify*

**fix**  $U \ U'$

**assume** *ne*: *inter-scheme*  $l \ U \neq \text{inter-scheme } l \ U'$  **and** *init*: *initial-segment* (*inter-scheme*  $l \ U$ ) (*inter-scheme*  $l \ U'$ )

**assume** *Form*  $l \ U$

**then obtain**  $kp \ kq \ xs \ ys$  **where**  $l = kp+kq - 1 \ U = \{xs, ys\}$

**and**  $U$ : *Form-Body*  $kp \ kq \ xs \ ys$  (*inter-scheme*  $l \ U$ ) **and**  $0 < kq \ kq \leq kp$   
 $kp \leq \text{Suc } kq$

**using** *assms* *inter-scheme* **by** *blast*

**then have**  $kp = ka \wedge kq = kb$

**using** *nz* **by** *linarith*

**then obtain**  $a \ as \ b \ bs \ c \ d$

**where** *len*:  $\text{length } (a\#as) = ka \ \text{length } (b\#bs) = kb$

**and**  $c$ :  $c = \text{acc-lengths } 0 \ (a\#as)$

**and**  $d$ :  $d = \text{acc-lengths } 0 \ (b\#bs)$

**and**  $Ueq$ : *inter-scheme*  $l \ U = \text{concat } [c, a, d, b] \ @ \ \text{interact } as \ bs$

**using**  $U$  **by** (*auto simp: Form-Body.simps*)

**assume** *Form*  $l \ U'$

**then obtain**  $kp' \ kq' \ xs' \ ys'$  **where**  $l = kp'+kq' - 1 \ U' = \{xs', ys'\}$

**and**  $U'$ : *Form-Body*  $kp' \ kq' \ xs' \ ys'$  (*inter-scheme*  $l \ U'$ ) **and**  $0 < kq' \ kq' \leq kp' \ kq' \leq \text{Suc } kq'$

**using** *assms* *inter-scheme* **by** *blast*

**then have**  $kp' = ka \wedge kq' = kb$

**using** *nz* **by** *linarith*

**then obtain**  $a' \ as' \ b' \ bs' \ c' \ d'$

**where** *len'*:  $\text{length } (a'\#as') = ka \ \text{length } (b'\#bs') = kb$

**and**  $c'$ :  $c' = \text{acc-lengths } 0 \ (a'\#as')$

**and**  $d'$ :  $d' = \text{acc-lengths } 0 \ (b'\#bs')$

**and**  $Ueq'$ : *inter-scheme*  $l \ U' = \text{concat } [c', a', d', b'] \ @ \ \text{interact } as' \ bs'$

```

    using U' by (auto simp: Form-Body.simps)
  have [simp]: length bs' = length bs length as' = length as
    using len len' by auto
  have inter-scheme l U ≠ [] inter-scheme l U' ≠ []
    using Form-Body-nonempty U U' by auto
  define u1 where u1 ≡ hd (inter-scheme l U)
  have u1-eq': u1 = hd (inter-scheme l U')
    using ⟨inter-scheme l U ≠ []⟩ init u1-def initial-segment-ne by fastforce
  have au1: u1 = length a
    by (simp add: u1-def Ueq c)
  have au1': u1 = length a'
    by (simp add: u1-eq' Ueq' c')
  have len-eqk: length c' = ka length d' = kb length c' = ka length d' = kb
    using c d len c' d' len' by auto
  have take: take (ka + u1 + kb) (c @ a @ d @ l) = c @ a @ d
    take (ka + u1 + kb) (c' @ a' @ d' @ l) = c' @ a' @ d' for l
    using c d c' d' len by (simp-all flip: au1 au1')
  have leU: ka + u1 + kb ≤ length (inter-scheme l U)
    using c d len by (simp add: au1 Ueq)
  then have take (ka + u1 + kb) (inter-scheme l U) = take (ka + u1 + kb)
(inter-scheme l U')
    using take-initial-segment init by blast
  then have §: c @ a @ d = c' @ a' @ d'
    by (metis Ueq Ueq' append.assoc concat.simps(2) take)
  have length (inter-scheme l U) = ka + (c @ a @ d)!(ka-1) + kb + last d
    by (simp add: Ueq c d length-interact nth-append flip: len)
  moreover have length (inter-scheme l U') = ka + (c' @ a' @ d')!(ka-1) +
kb + last d'
    by (simp add: Ueq' c' d' length-interact nth-append flip: len')
  moreover have last d = last d'
    using § c d d' len'(1) len-eqk(1) by auto
  ultimately have length (inter-scheme l U) = length (inter-scheme l U')
    by (simp add: §)
  then show False
    using init initial-segment-length-eq ne by blast
qed
qed

```

### 3.9 Larson's Lemma 3.6

**proposition lemma-3-6:**

**fixes**  $g :: \text{nat list set} \Rightarrow \text{nat}$

**assumes**  $g: g \in [WW]^2 \rightarrow \{0,1\}$

**obtains**  $N j$  **where** *infinite*  $N$

**and**  $\bigwedge k u. \llbracket k > 0; u \in [WW]^2; \text{Form } k u; [\text{enum } N k] < \text{inter-scheme } k u; \text{List.set } (\text{inter-scheme } k u) \subseteq N \rrbracket \Longrightarrow g u = j k$

**proof** –

**define**  $\Phi$  **where**  $\Phi \equiv \lambda m::\text{nat}. \lambda M. \text{infinite } M \wedge m < \text{Inf } M$

**define**  $\Psi$  **where**  $\Psi \equiv \lambda l m n::\text{nat}. \lambda M N j. n > m \wedge N \subseteq M \wedge n \in M$

$\wedge (\forall U. \text{Form } l \ U \wedge U \subseteq WW \wedge [n] < \text{inter-scheme } l \ U \wedge \text{list.set } (\text{inter-scheme } l \ U) \subseteq N \longrightarrow g \ U = j)$   
**have** \*:  $\exists n \ N \ j. \Phi \ n \ N \wedge \Psi \ l \ m \ n \ M \ N \ j$  **if**  $l > 0$   $\Phi \ m \ M$  **for**  $l \ m :: \text{nat}$  **and**  $M :: \text{nat set}$   
**proof** –  
**define**  $FF$  **where**  $FF \equiv \{U \in [WW]^2. \text{Form } l \ U\}$   
**define**  $h$  **where**  $h \equiv \lambda z s. g \ (\text{inv-into } FF \ (\text{inter-scheme } l) \ z s)$   
**have**  $\text{thin} \ (\text{inter-scheme } l \ ' \ FF)$   
**using**  $\langle l > 0 \rangle$   $\text{lemma-3-11}$  **by**  $(\text{simp add: thin-def } FF\text{-def})$   
**moreover**  
**have**  $\text{inter-scheme } l \ ' \ FF \subseteq WW$   
**using**  $\text{inter-scheme-simple } \langle 0 < l \rangle$   $FF\text{-def}$  **by**  $\text{blast}$   
**moreover**  
**have**  $h \ ' \ \{xs \in \text{inter-scheme } l \ ' \ FF. \text{List.set } xs \subseteq M\} \subseteq \{..<2\}$   
**using**  $g \ \text{inv-into-into}[\text{of concl: } FF \ \text{inter-scheme } l]$   
**by**  $(\text{force simp: } h\text{-def } FF\text{-def } Pi\text{-iff})$   
**ultimately**  
**obtain**  $j \ N$  **where**  $j < 2$   $\text{infinite } N \ N \subseteq M$  **and**  $h j: h \ ' \ \{xs \in \text{inter-scheme } l \ ' \ FF. \text{List.set } xs \subseteq N\} \subseteq \{j\}$   
**using**  $\langle \Phi \ m \ M \rangle$  **unfolding**  $\Phi\text{-def}$  **by**  $(\text{blast intro: Nash-Williams-WW } [\text{of } M])$   
**let**  $?n = \text{Inf } N$   
**have**  $?n > m$   
**using**  $\langle \Phi \ m \ M \rangle$   $\langle \text{infinite } N \rangle$  **unfolding**  $\Phi\text{-def}$   $\text{Inf-nat-def}$   $\text{infinite-nat-iff-unbounded}$   
**by**  $(\text{metis LeastI-ex } \langle N \subseteq M \rangle \text{le-less-trans not-less not-less-Least subsetD})$   
**have**  $g \ U = j$  **if**  $\text{Form } l \ U \ U \subseteq WW \ [?n] < \text{inter-scheme } l \ U \ \text{list.set } (\text{inter-scheme } l \ U) \subseteq N - \{?n\}$  **for**  $U$   
**proof** –  
**obtain**  $xs \ ys$  **where**  $xy s: xs \neq ys \ U = \{xs, ys\}$   
**using**  $\text{Form-elim-upair } \langle \text{Form } l \ U \rangle$  **by**  $\text{blast}$   
**moreover** **have**  $\text{inj-on} \ (\text{inter-scheme } l) \ FF$   
**using**  $\langle 0 < l \rangle$   $\text{inj-on-def}$   $\text{inter-scheme-injective } FF\text{-def}$  **by**  $\text{blast}$   
**moreover**  
**have**  $g \ (\text{inv-into } FF \ (\text{inter-scheme } l) \ (\text{inter-scheme } l \ U)) = j$   
**using**  $h j$  **that**  $xy s \ \text{subset-Diff-insert}$  **by**  $(\text{fastforce simp: } h\text{-def } FF\text{-def } \text{image-iff})$   
**ultimately show**  $?thesis$   
**using**  $\text{that } FF\text{-def}$  **by**  $\text{auto}$   
**qed**  
**moreover** **have**  $?n < \text{Inf} \ (N - \{?n\})$   
**by**  $(\text{metis Diff-iff Inf-nat-def Inf-nat-def1 } \langle \text{infinite } N \rangle \text{finite.emptyI infinite-remove linorder-neqE-nat not-less-Least singletonI})$   
**moreover** **have**  $?n \in M$   
**by**  $(\text{metis Inf-nat-def1 } \langle N \subseteq M \rangle \langle \text{infinite } N \rangle \text{finite.emptyI subsetD})$   
**ultimately** **have**  $\Phi \ ?n \ (N - \{?n\}) \wedge \Psi \ l \ m \ ?n \ M \ (N - \{?n\}) \ j$   
**using**  $\langle \Phi \ m \ M \rangle$   $\langle \text{infinite } N \rangle$   $\langle N \subseteq M \rangle$   $\langle ?n > m \rangle$  **by**  $(\text{auto simp: } \Phi\text{-def } \Psi\text{-def})$   
**then show**  $?thesis$   
**by**  $\text{blast}$   
**qed**  
**have**  $\text{base: } \Phi \ 0 \ \{0 < ..\}$

**unfolding**  $\Phi$ -def by (metis infinite-Ioi Inf-nat-def1 greaterThan-iff greaterThan-non-empty)  
**have** step: Ex  $(\lambda(n,N,j). \Phi n N \wedge \Psi l m n M N j)$  **if**  $\Phi m M l > 0$  **for**  $m M l$   
**using** \* [of  $l m M$ ] **that by** (auto simp:  $\Phi$ -def)  
**define**  $G$  **where**  $G \equiv \lambda l m M. @ (n,N,j). \Phi n N \wedge \Psi (Suc l) m n M N j$   
**have**  $G\Phi$ :  $(\lambda(n,N,j). \Phi n N) (G l m M)$  **and**  $G\Psi$ :  $(\lambda(n,N,j). \Psi (Suc l) m n M N j) (G l m M)$   
**if**  $\Phi m M$  **for**  $l m M$   
**using** step [OF that, of  $Suc l$ ] **by** (force simp:  $G$ -def dest: some-eq-imp)+  
**have**  $G$ -increasing:  $(\lambda(n,N,j). n > m \wedge N \subseteq M \wedge n \in M) (G l m M)$  **if**  $\Phi m M$  **for**  $l m M$   
**using**  $G\Psi$  [OF that, of  $l$ ] **that by** (simp add:  $\Psi$ -def split: prod.split-asm)  
**define**  $H$  **where**  $H \equiv rec\text{-nat } (0, \{0<..\}, 0) (\lambda l (m,M,j). G l m M)$   
**have**  $H$ -simps:  $H 0 = (0, \{0<..\}, 0) \wedge l. H (Suc l) = (case H l of (m,M,j) \Rightarrow G l m M)$   
**by** (simp-all add:  $H$ -def)  
**have**  $H\Phi$ :  $(\lambda(n,N,j). \Phi n N) (H l)$  **for**  $l$   
**by** (induction  $l$ ) (use base  $G\Phi$  **in**  $\langle$ auto simp:  $H$ -simps split: prod.split-asm $\rangle$ )  
**define**  $\nu$  **where**  $\nu \equiv (\lambda l. case H l of (n,M,j) \Rightarrow n)$   
**have**  $H$ -inc:  $\nu l \geq l$  **for**  $l$   
**proof** (induction  $l$ )  
**case** ( $Suc l$ )  
**then show** ?case  
**using**  $H\Phi$  [of  $l$ ]  $G$ -increasing [of  $\nu l$ ]  
**apply** (clarsimp simp:  $H$ -simps  $\nu$ -def split: prod.split)  
**by** (metis (no-types, lifting) case-prodD leD le-less-trans not-less-eq-eq)  
**qed** auto  
**let** ? $N = range \nu$   
**define**  $j$  **where**  $j \equiv \lambda l. case H l of (n,M,j) \Rightarrow j$   
**have**  $H$ -increasing-Suc:  $(case H k of (n, N, j') \Rightarrow N) \supseteq (case H (Suc k) of (n, N, j') \Rightarrow insert n N)$  **for**  $k$   
**using**  $H\Phi$  [of  $k$ ]  
**by** (force simp:  $H$ -simps split: prod.split dest:  $G$ -increasing [**where**  $l=k$ ])  
**have**  $H$ -increasing-superset:  $(case H k of (n, N, j') \Rightarrow N) \supseteq (case H (n+k) of (n, N, j') \Rightarrow N)$  **for**  $k n$   
**proof** (induction  $n$ )  
**case** ( $Suc n$ )  
**then show** ?case  
**using**  $H$ -increasing-Suc [of  $n+k$ ] **by** (auto split: prod.split-asm)  
**qed** auto  
**then have**  $H$ -increasing-less:  $(case H k of (n, N, j') \Rightarrow N) \supseteq (case H l of (n, N, j') \Rightarrow insert n N)$   
**if**  $k < l$  **for**  $k l$   
**by** (smt (verit, best)  $H$ -increasing-Suc add.commute less-natE order-trans that)  
**have**  $\nu k < \nu (Suc k)$  **for**  $k$   
**using**  $H\Phi$  [of  $k$ ] **unfolding**  $\nu$ -def  
**by** (auto simp:  $H$ -simps split: prod.split dest:  $G$ -increasing [**where**  $l=k$ ])  
**then have** strict-mono- $\nu$ : strict-mono  $\nu$   
**by** (simp add: strict-mono-Suc-iff)  
**then have** enum- $N$ : enum ? $N = \nu$

```

  by (metis enum-works nat-infinite-iff range-strict-mono-ext)
have **:  $?N \cap \{n < ..\} \subseteq N'$  if  $H: H k = (n, N', j)$  for  $n N' k j$ 
proof clarify
  fix l
  assume  $n < \nu l$ 
  then have  $False$  if  $l \leq k$ 
    using that strict-monoD [OF strict-mono- $\nu$ , of l k] H by (force simp:  $\nu$ -def)
  then have  $k < l$  using not-less by blast
  then obtain M j where Mj:  $H l = (\nu l, M, j)$ 
    unfolding  $\nu$ -def
    by (metis (mono-tags, lifting) case-prod-conv old.prod.exhaust)
  then show  $\nu l \in N'$ 
    using that H-increasing-less [OF  $\langle k < l \rangle$ ] Mj by auto
qed
show thesis
proof
  show infinite ( $?N::nat$  set)
    using H-inc infinite-nat-iff-unbounded-le by auto
next
fix l U
assume  $0 < l$  and  $U: U \in [WW]^2$ 
  and interU:  $[enum ?N l] < inter\ scheme\ l\ U\ Form\ l\ U$ 
  and sub:  $list.set (inter\ scheme\ l\ U) \subseteq ?N$ 
obtain k where k:  $l = Suc\ k$ 
  using  $\langle 0 < l \rangle$  gr0-conv-Suc by blast
have  $g\ U = v$  if  $H\ k = (m, M, j0)$  and  $G\ k\ m\ M = (n, N', v)$ 
  for  $m\ M\ j0\ n\ N'\ v$ 
proof -
  have  $n: \nu (Suc\ k) = n$ 
    using that by (simp add:  $\nu$ -def H-simps)
  have  $\{..enum (range\ \nu)\ l\} \cap list.set (inter\ scheme\ l\ U) = \{\}$ 
  using inter-scheme-strict-sorted  $\langle 0 < l \rangle$  interU singleton-less-list-iff strict-sorted-iff
by blast
  then have  $list.set (inter\ scheme (Suc\ k)\ U) \subseteq N'$ 
    using that sub ** [of Suc k n N' v] Suc-le-eq not-less-eq-eq
    by (fastforce simp: k n enum-N H-simps)
  then show ?thesis
    using that interU U G $\Psi$  [of m M k] H $\Phi$  [of k]
    by (auto simp:  $\Psi$ -def k enum-N H-simps n nsets-def)
qed
with U show  $g\ U = j\ l$ 
  by (auto simp: k j-def H-simps split: prod.split)
qed
qed

```

### 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**  $nsets\ A\ 2 = \{\{x,y\} \mid x\ y.\ x \in A \wedge y \in A \wedge (x,y) \in r\}$   
*(is - = ?rhs)*  
**proof**  
**show**  $nsets\ A\ 2 \subseteq ?rhs$   
**using** *tot*  
**unfolding** *numeral-nat total-on-def nsets-def*  
**by** (*fastforce simp: card-Suc-eq Set.doubleton-eq-iff not-less*)  
**show**  $?rhs \subseteq nsets\ A\ 2$   
**using** *irr unfolding numeral-nat* **by** (*force simp: nsets-def card-Suc-eq irrefl-def*)  
**qed**

**lemma** *lenlex-nsets-2-eq*:  $nsets\ A\ 2 = \{\{x,y\} \mid x\ y.\ x \in A \wedge y \in A \wedge (x,y) \in lenlex\ less-than\ \}$   
**using** *total-order-nsets-2-eq* **by** (*simp add: total-order-nsets-2-eq irrefl-def*)

**lemma** *sum-sorted-list-of-set-map*:  $finite\ I \implies sum-list\ (map\ f\ (list-of\ I)) = sum\ f\ I$   
**proof** (*induction card I arbitrary: I*)  
**case** (*Suc n I*)  
**then have** [*simp*]:  $I \neq \{\}$   
**by** *auto*  
**have**  $sum-list\ (map\ f\ (list-of\ (I - \{Min\ I\}))) = sum\ f\ (I - \{Min\ I\})$   
**using** *Suc* **by** *auto*  
**then show** *?case*  
**using** *Suc.prem1 sum.remove [of I Min I f]*  
**by** (*simp add: sorted-list-of-set-nonempty Suc*)  
**qed** *auto*

**lemma** *sorted-list-of-set-UN-eq-concat*:  
**assumes** *I*: *strict-mono-sets I f* *finite I* **and** *fin*:  $\bigwedge i.\ finite\ (f\ i)$   
**shows**  $list-of\ (\bigcup i \in I.\ f\ i) = concat\ (map\ (list-of\ \circ\ f)\ (list-of\ I))$   
**using** *I*  
**proof** (*induction card I arbitrary: I*)  
**case** (*Suc n I*)  
**then have**  $I \neq \{\}$  **and** *Iexp*:  $I = insert\ (Min\ I)\ (I - \{Min\ I\})$   
**using** *Min-in Suc.hyps(2) Suc.prem1(2)* **by** *fastforce+*  
**have** *IH*:  $list-of\ (\bigcup (f\ ' (I - \{Min\ I\}))) = concat\ (map\ (list-of\ \circ\ f)\ (list-of\ (I - \{Min\ I\})))$   
**using** *Suc unfolding strict-mono-sets-def*  
**by** (*metis DiffE Iexp card-Diff-singleton diff-Suc-1 finite-Diff insertI1*)  
**have**  $list-of\ (\bigcup (f\ ' I)) = list-of\ (\bigcup (f\ ' (insert\ (Min\ I)\ (I - \{Min\ I\}))))$   
**using** *Iexp* **by** *auto*  
**also have**  $\dots = list-of\ (f\ (Min\ I) \cup \bigcup (f\ ' (I - \{Min\ I\})))$   
**by** (*metis Union-image-insert*)  
**also have**  $\dots = list-of\ (f\ (Min\ I)) @ list-of\ (\bigcup (f\ ' (I - \{Min\ I\})))$

```

proof (rule sorted-list-of-set-Un)
  show  $f (Min I) \ll \bigcup (f \text{ ` } (I - \{Min I\}))$ 
    using Suc.prems  $\langle I \neq \{\} \rangle$  strict-mono-less-sets-Min by blast
  show finite  $(\bigcup (f \text{ ` } (I - \{Min I\})))$ 
    by (simp add:  $\langle \text{finite } I \rangle$  fin)
  qed (use fin in auto)
  also have  $\dots = \text{list-of } (f (Min I)) @ \text{concat } (\text{map } (\text{list-of } \circ f) (\text{list-of } (I - \{Min I\})))$ 
    using IH by metis
  also have  $\dots = \text{concat } (\text{map } (\text{list-of } \circ f) (\text{list-of } I))$ 
    by (simp add: Suc.prems(2)  $\langle I \neq \{\} \rangle$  sorted-list-of-set-nonempty)
  finally show ?case .
qed auto

```

### 3.10.2 Lemma 3.7 of Jean A. Larson, *ibid*.

**proposition** *lemma-3-7*:

**assumes** *infinite*  $N$   $l > 0$

**obtains**  $M$  **where**  $M \in [WW]^m$

$$\bigwedge U. U \in [M]^2 \implies \text{Form } l U \wedge \text{List.set } (\text{inter-scheme } l U) \subseteq N$$

**proof** (*cases*  $m < 2$ )

**case** *True*

**obtain**  $w$  **where**  $w \in WW$

**using** *WW-def strict-sorted-into-WW* **by** *auto*

**define**  $M$  **where**  $M \equiv \text{if } m=0 \text{ then } \{\} \text{ else } \{w\}$

**have**  $M: M \in [WW]^m$

**using** *True* **by** (*auto simp*: *M-def nsets-def*  $w$ )

**have** [*simp*]:  $[M]^2 = \{\}$

**using** *True* **by** (*auto simp*: *M-def nsets-def*  $w$  *dest*: *subset-singletonD*)

**show** ?*thesis*

**using**  $M$  **that** **by** *fastforce*

**next**

**case** *False*

**then have**  $m \geq 2$

**by** *auto*

**have** *nonz*:  $(\text{enum } N \circ \text{Suc}) i > 0$  **for**  $i$

**using** *assms*(1) *le-enumerate less-le-trans* **by** *fastforce*

**note** *infinite-nxt-N* = *infinite-nxtN* [*OF*  $\langle \text{infinite } N \rangle$ , *iff*]

**note**  $\langle \text{infinite } N \rangle$  [*iff*]

**have** [*simp*]:  $\{n <..< \text{Suc } n\} = \{\}$   $\{..< 1::\text{nat}\} = \{0\}$  **for**  $n$

**by** *auto*

**note** *One-nat-def* [*simp del*]

**define** *DF-Suc* **where**  $\text{DF-Suc} \equiv \lambda k D. \text{enum } (\text{nxt } N (\text{enum } (\text{nxt } N (\text{Max } D)) (\text{Inf } D - 1))) \text{ ` } \{..< \text{Suc } k\}$

**define** *DF* **where**  $\text{DF} \equiv \lambda k n. (\text{DF-Suc } k \sim n) ((\text{enum } N \circ \text{Suc}) \text{ ` } \{..< \text{Suc } k\})$

**have** *DF-simps*:  $\text{DF } k 0 = (\text{enum } N \circ \text{Suc}) \text{ ` } \{..< \text{Suc } k\}$   $\text{DF } k (\text{Suc } i) = \text{DF-Suc } k (\text{DF } k i)$  **for**  $i k$

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

```

have card-DF:  $\text{card } (DF\ k\ i) = \text{Suc } k$  for  $i\ k$ 
proof (induction i)
  case 0
  have inj-on ( $\text{enum } N \circ \text{Suc}$ )  $\{..<\text{Suc } k\}$ 
    by (simp add: assms(1) strict-mono-def strict-mono-imp-inj-on)
  with 0 show ?case
    using card-image DF-simps by fastforce
next
  case ( $\text{Suc } i$ )
  then show ?case
    by (simp add: <infinite N> DF-simps DF-Suc-def card-image infinite-nxtN
strict-mono-enum strict-mono-imp-inj-on)
qed
have DF-ne:  $DF\ k\ i \neq \{\}$  for  $i\ k$ 
  by (metis card-DF card-lessThan lessThan-empty-iff nat.simps(3))

have finite-DF:  $\text{finite } (DF\ k\ i)$  for  $i\ k$ 
  by (induction i) (auto simp: DF-simps DF-Suc-def)
have DF-Suc:  $DF\ k\ i \ll DF\ k\ (\text{Suc } i)$  for  $i\ k$ 
  unfolding less-sets-def
  by (force simp: finite-DF DF-simps DF-Suc-def
    intro!: greaterThan-less-enum nxt-subset-greaterThan atLeast-le-enum nxt-subset-atLeast
infinite-nxtN [OF <infinite N>])
have DF-DF:  $DF\ k\ i \ll DF\ k\ j$  if  $i < j$  for  $i\ j\ k$ 
  by (meson DF-Suc DF-ne UNIV-I less-sets-imp-strict-mono-sets strict-mono-setsD
that)
then have sm-DF: strict-mono-sets UNIV ( $DF\ k$ ) for  $k$ 
  by (simp add: strict-mono-sets-def)

have DF-gt0:  $0 < \text{Inf } (DF\ k\ i)$  for  $i\ k$ 
proof (cases i)
  case 0
  then show ?thesis
    by (metis DF-ne DF-simps(1) Inf-nat-def1 imageE nonz)
next
  case ( $\text{Suc } n$ )
  then show ?thesis
    by (metis DF-Suc DF-ne Inf-nat-def1 grOI gr-implies-not0 less-sets-def)
qed
have DF-N:  $DF\ k\ i \subseteq N$  for  $i\ k$ 
proof (induction i)
  case 0
  then show ?case
    using  $\langle \text{infinite } N \rangle$  range-enum by (auto simp: DF-simps)
next
  case ( $\text{Suc } i$ )
  then show ?case
    unfolding DF-simps DF-Suc-def image-subset-iff

```



by (metis IntE <infinite N> enumerate-in-set infinite-nxtN nxt-def)  
 qed

have sm-enum-DF: strict-mono-on {..k} (enum (DF k i)) for k i  
 by (metis card-DF enum-works-finite finite-DF lessThan-Suc-atMost)

define AF where AF  $\equiv \lambda k i. \text{enum } (nxt N (Max (DF k i))) \text{ ' } \{..<Inf (DF k i)\}$   
 have AF-ne: AF k i  $\neq \{\}$  for i k  
 by (auto simp: AF-def lessThan-empty-iff DF-gt0)  
 have finite-AF [simp]: finite (AF k i) for i k  
 by (simp add: AF-def)  
 have card-AF: card (AF k i) =  $\prod$  (DF k i) for k i  
 by (simp add: AF-def card-image inj-enum-nxt)

have DF-AF: DF k i  $\ll$  AF k i for i k  
 unfolding less-sets-def AF-def  
 by (simp add: finite-DF greaterThan-less-enum nxt-subset-greaterThan)

have E:  $[x \leq y; \text{infinite } M] \implies \text{enum } M x < \text{enum } (nxt N (\text{enum } M y)) z$  for  
 x y z M  
 by (metis infinite-nxt-N dual-order.eq-iff enumerate-mono greaterThan-less-enum  
 nat-less-le nxt-subset-greaterThan)

have AF-DF-Suc: AF k i  $\ll$  DF k (Suc i) for i k  
 by (auto simp: DF-simps DF-Suc-def less-sets-def AF-def E)

have AF-DF: AF k p  $\ll$  DF k q if p < q for k p q  
 by (metis AF-DF-Suc DF-ne Suc-lessI UNIV-I less-sets-trans sm-DF strict-mono-sets-def  
 that)

have AF-Suc: AF k i  $\ll$  AF k (Suc i) for i k  
 using AF-DF-Suc DF-AF DF-ne less-sets-trans by blast  
 then have sm-AF: strict-mono-sets UNIV (AF k) for k  
 by (simp add: AF-ne less-sets-imp-strict-mono-sets)

define del where del  $\equiv \lambda k i j. \text{enum } (DF k i) j - \text{enum } (DF k i) (j - 1)$

define QF where QF k  $\equiv \text{wfrec pair-less } (\lambda f (j, i). \text{if } j=0 \text{ then } AF k i \text{ else let } r = (\text{if } i=0 \text{ then } f (j-1, m-1) \text{ else } f (j, i-1)) \text{ in } \text{enum } (nxt N (Suc (Max r))) \text{ ' } \{..< del k (\text{if } j=k \text{ then } m - Suc i \text{ else } i) j\})$   
 for k  
 note cut-apply [simp]

have finite-QF [simp]: finite (QF k p) for p k  
 using wf-pair-less  
 proof (induction p rule: wf-induct-rule)

```

    case (less p)
  then show ?case
    by (simp add: def-wfrec [OF QF-def, of k p] split: prod.split)
qed

have del-gt-0:  $\llbracket j < \text{Suc } k; 0 < j \rrbracket \implies 0 < \text{del } k \ i \ j$  for  $i \ j \ k$ 
  by (simp add: card-DF del-def finite-DF)

have QF-ne [simp]:  $QF \ k \ (j, i) \neq \{\}$  if  $j: j < \text{Suc } k$  for  $j \ i \ k$ 
  using wf-pair-less j
proof (induction (j,i) rule: wf-induct-rule)
  case less
  then show ?case
    by (auto simp: def-wfrec [OF QF-def, of k (j,i)] AF-ne lessThan-empty-iff
del-gt-0)
qed

have QF-0 [simp]:  $QF \ k \ (0, i) = AF \ k \ i$  for  $i \ k$ 
  by (simp add: def-wfrec [OF QF-def])

have QF-Suc:  $QF \ k \ (\text{Suc } j, 0) = \text{enum } (\text{nxt } N \ (\text{Suc } (\text{Max } (QF \ k \ (j, m - 1))))$ 
  ‘  $\{.. < \text{del } k \ (\text{if } \text{Suc } j = k \ \text{then } m - 1 \ \text{else } 0) \ (\text{Suc } j)\}$  for  $j \ k$ 
  apply (simp add: def-wfrec [OF QF-def, of k (Suc j, 0)] One-nat-def)
  apply (simp add: pair-less-def cut-def)
  done

have QF-Suc-Suc:  $QF \ k \ (\text{Suc } j, \text{Suc } i)$ 
  =  $\text{enum } (\text{nxt } N \ (\text{Suc } (\text{Max } (QF \ k \ (\text{Suc } j, i))))$  ‘  $\{.. < \text{del } k \ (\text{if } \text{Suc}$ 
 $j = k \ \text{then } m - \text{Suc}(\text{Suc } i) \ \text{else } \text{Suc } i) \ (\text{Suc } j)\}$ 
  for  $i \ j \ k$ 
  by (simp add: def-wfrec [OF QF-def, of k (Suc j, Suc i)])

have less-QF1:  $QF \ k \ (j, m - 1) \ll QF \ k \ (\text{Suc } j, 0)$  for  $j \ k$ 
  by (auto simp: def-wfrec [OF QF-def, of k (Suc j, 0)] pair-lessI1 enum-nxt-ge
  intro!: less-sets-weaken2 [OF less-sets-Suc-Max])

have less-QF2:  $QF \ k \ (j, i) \ll QF \ k \ (j, \text{Suc } i)$  for  $j \ i \ k$ 
  by (auto simp: def-wfrec [OF QF-def, of k (j, Suc i)] pair-lessI2 enum-nxt-ge
  intro: less-sets-weaken2 [OF less-sets-Suc-Max] strict-mono-setsD [OF
sm-AF])

have less-QF-same:  $QF \ k \ (j, i') \ll QF \ k \ (j, i)$ 
  if  $i' < i \leq k$  for  $i' \ i \ j \ k$ 
proof (rule strict-mono-setsD [OF less-sets-imp-strict-mono-sets])
  show  $QF \ k \ (j, i) \ll QF \ k \ (j, \text{Suc } i)$  for  $i$ 
    by (simp add: less-QF2)
  show  $QF \ k \ (j, i) \neq \{\}$  if  $0 < i$  for  $i$ 
    using that by (simp add:  $\langle j \leq k \rangle \text{le-imp-less-Suc}$ )

```

qed (use that in auto)

have less-QF-step:  $QF\ k\ (j-1, i') \ll QF\ k\ (j, i)$   
 if  $0 < j\ j \leq k\ i' < m$  for  $j\ i'\ i\ k$

proof -  
 have less-QF1':  $QF\ k\ (j-1, m-1) \ll QF\ k\ (j, 0)$  if  $j > 0$  for  $j$   
 by (metis less-QF1 that Suc-pred One-nat-def)  
 have  $QF\ k\ (j-1, i') \ll QF\ k\ (j, 0)$   
 proof (cases  $i' = m - 1$ )  
 case True  
 then show ?thesis  
 using less-QF1'  $\langle 0 < j \rangle$  by blast  
 next  
 case False  
 show ?thesis  
 using False that less-sets-trans [OF less-QF-same less-QF1' QF-ne] by auto  
 qed  
 then show ?thesis  
 by (metis QF-ne less-QF-same less-Suc-eq-le less-sets-trans  $\langle j \leq k \rangle$  zero-less-iff-neq-zero)  
 qed

have less-QF:  $QF\ k\ (j', i') \ll QF\ k\ (j, i)$   
 if  $j: j' < j\ j \leq k$  and  $i: i' < m\ i < m$  for  $j'\ j\ i'\ i\ k$   
 using  $j$

proof (induction  $j-j'$  arbitrary:  $j$ )  
 case (Suc d)  
 show ?case  
 proof (cases  $j' < j - 1$ )  
 case True  
 then have  $QF\ k\ (j', i') \ll QF\ k\ (j - 1, i)$   
 using Suc.hyps Suc.prem(2) by force  
 then show ?thesis  
 by (rule less-sets-trans [OF less-QF-step QF-ne]) (use Suc i in auto)  
 next  
 case False  
 then have  $j' = j - 1$   
 using  $\langle j' < j \rangle$  by linarith  
 then show ?thesis  
 using Suc.hyps  $\langle j \leq k \rangle$  less-QF-step i by auto  
 qed  
 qed auto

have sm-QF: strict-mono-sets  $(\{..k\} \times \{..<m\})$  (QF k) for k  
 unfolding strict-mono-sets-def  
 proof (intro strip)  
 fix p q  
 assume  $p: p \in \{..k\} \times \{..<m\}$  and  $q: q \in \{..k\} \times \{..<m\}$  and  $p < q$   
 then obtain  $j'\ i'\ j\ i$  where  $\S: p = (j', i')\ q = (j, i)\ i' < m\ i < m\ j' \leq k\ j \leq k$   
 using surj-pair [of p] surj-pair [of q] by blast

**with**  $\langle p < q \rangle$  **have**  $j' < j \vee j' = j \wedge i' < i$   
**by** *auto*  
**then show**  $QF\ k\ p \ll QF\ k\ q$   
**using** § *less-QF less-QF-same* **by** *presburger*  
**qed**  
**then have** *sm-QF1: strict-mono-sets*  $\{..<ka\}$   $(\lambda j. QF\ k\ (j,i))$   
**if**  $i < m$  **Suc**  $k \geq ka$   $ka \geq k$  **for**  $ka\ k\ i$   
**proof** –  
**have**  $\{..<ka\} \subseteq \{..k\}$   
**by** (*metis lessThan-Suc-atMost lessThan-subset-iff*  $\langle Suc\ k \geq ka \rangle$ )  
**then show** *?thesis*  
**by** (*simp add: less-QF strict-mono-sets-def subset-iff that*)  
**qed**  
  
**have** *disjoint-QF:  $i'=i \wedge j'=j$  if  $\neg$  disjoint*  $(QF\ k\ (j', i')) (QF\ k\ (j, i))$   $j' \leq k\ j \leq$   
 $k\ i' < m\ i < m$  **for**  $i'\ i\ j'\ j\ k$   
**using** *that strict-mono-sets-imp-disjoint* [*OF sm-QF*]  
**by** (*force simp: pairwise-def*)  
  
**have** *card-QF: card*  $(QF\ k\ (j, i)) = (if\ j=0\ then\ \square\ (DF\ k\ i)\ else\ del\ k\ (if\ j = k$   
*then*  $m - Suc\ i\ else\ i)\ j)$   
**for**  $i\ k\ j$   
**proof** (*cases j*)  
**case**  $0$   
**then show** *?thesis*  
**by** (*simp add: AF-def card-image inj-enum-nxt*)  
**next**  
**case**  $(Suc\ j')$   
**show** *?thesis*  
**by** (*cases i; simp add: Suc One-nat-def QF-Suc QF-Suc-Suc card-image*  
*inj-enum-nxt*)  
**qed**  
**have** *AF-non-Nil: list-of*  $(AF\ k\ i) \neq \square$  **for**  $k\ i$   
**by** (*simp add: AF-ne*)  
**have** *QF-non-Nil: list-of*  $(QF\ k\ (j, i)) \neq \square$  **if**  $j < Suc\ k$  **for**  $i\ j\ k$   
**by** (*simp add: that*)  
  
**have** *AF-subset-N: AF*  $k\ i \subseteq N$  **for**  $i\ k$   
**unfolding** *AF-def image-subset-iff*  
**using** *nxt-subset enumerate-in-set infinite-nxtN*  $\langle infinite\ N \rangle$  **by** *blast*  
  
**have** *QF-subset-N: QF*  $k\ (j, i) \subseteq N$  **for**  $i\ j\ k$   
**proof** (*induction j*)  
**case**  $(Suc\ j)$   
**show** *?case*  
**by** (*cases i*) (*use nxt-subset enumerate-in-set in*  $\langle (force\ simp: QF-Suc\ QF-Suc-Suc) + \rangle$ )  
**qed** (*use AF-subset-N in auto*)  
  
**obtain**  $ka\ k$  **where**  $k > 0$  **and**  $kka: k \leq ka\ ka \leq Suc\ k\ l = ((ka+k) - 1)$

```

  by (metis One-nat-def assms(2) diff-add-inverse form-cases le0 le-refl)
then have ka > 0
  using dual-order.strict-trans1 by blast
have ka-k-or-Suc: ka = k ∨ ka = Suc k
  using kka by linarith
have lessThan-k: {..

```

**have**  $ka - 1 \leq k$   
**using**  $kka(2)$  **by** *linarith*  
**then have** *sum-card-RF'* [*simp*]:  
 $(\sum j < ka. \text{card } (RF\ j\ i)) = \text{enum } (DF\ k\ i)\ (ka - 1)$  **if**  $i < m$  **for**  $i$   
**using** *sum-card-RF* [*of ka - 1 i*]  
**by** (*metis Suc-diff-1*  $\langle 0 < ka \rangle$  *lessThan-Suc-atMost* *that*)

**have** *enum-DF-le-iff* [*simp*]:  
 $\text{enum } (DF\ k\ i)\ j \leq \text{enum } (DF\ k\ i')\ j \iff i \leq i'$  (**is** *?lhs = -*)  
**if**  $j \leq k$  **for**  $i'\ i\ j\ k$   
**proof**  
**show**  $i \leq i'$  **if** *?lhs*  
**proof**  $-$   
**have**  $\text{enum } (DF\ k\ i)\ j \in DF\ k\ i$   
**by** (*simp add: card-DF finite-enumerate-in-set finite-DF le-imp-less-Suc*  $\langle j \leq k \rangle$ )  
**moreover have**  $\text{enum } (DF\ k\ i')\ j \in DF\ k\ i'$   
**by** (*simp add: card-DF finite-enumerate-in-set finite-DF le-imp-less-Suc* *that*)  
**ultimately have**  $\text{enum } (DF\ k\ i')\ j < \text{enum } (DF\ k\ i)\ j$  **if**  $i' < i$   
**using** *sm-DF* [*of k*] **by** (*meson UNIV-I less-sets-def strict-mono-setsD* *that*)  
**then show** *?thesis*  
**using** *not-less* **that by** *blast*  
**qed**  
**show** *?lhs* **if**  $i \leq i'$   
**by** (*metis DF-DF*  $\langle j \leq k \rangle$  *card-DF finite-DF finite-enumerate-in-set le-eq-less-or-eq* *less-Suc-eq-le less-sets-def* *that*)  
**qed**

**then have** *enum-DF-eq-iff* [*simp*]:  
 $\text{enum } (DF\ k\ i)\ j = \text{enum } (DF\ k\ i')\ j \iff i = i'$  **if**  $j \leq k$  **for**  $i'\ i\ j\ k$   
**by** (*metis le-antisym order-refl* *that*)  
**have** *enum-DF-less-iff* [*simp*]:  
 $\text{enum } (DF\ k\ i)\ j < \text{enum } (DF\ k\ i')\ j \iff i < i'$  **if**  $j \leq k$  **for**  $i'\ i\ j\ k$   
**by** (*meson enum-DF-le-iff not-less* *that*)

**have** *card-AF-sum*:  $\text{card } (AF\ k\ i) + (\sum j \in \{0 < .. < ka\}. \text{card } (RF\ j\ i)) = \text{enum } (DF\ k\ i)\ (ka - 1)$   
**if**  $i < m$  **for**  $i$   
**using** *that*  $\langle k > 0 \rangle$   $\langle k \leq ka \rangle$   $\langle ka \leq \text{Suc } k \rangle$   
**by** (*simp add: lessThan-k RF-0 flip: sum-card-RF'*)

**have** *sorted-list-of-set-iff* [*simp*]:  $\text{list-of } \{0 < .. < k\} = [] \iff k = 1$  **if**  $k > 0$  **for**  $k :: \text{nat}$   
**using** *atLeastSucLessThan-greaterThanLessThan* *that by* *fastforce*  
**show** *thesis*  $-$  *proof of main result*  
**proof**  
**have** *inj*: *inj-on*  $(\lambda i. \text{list-of } (\bigcup j < ka. RF\ j\ i))\ \{.. < m\}$   
**proof** (*clarsimp simp: inj-on-def*)

**fix**  $x y$   
**assume**  $x < m y < m$  *list-of*  $(\bigcup_{j < ka}. RF\ j\ x) = \text{list-of } (\bigcup_{j < ka}. RF\ j\ y)$   
**then have**  $eq: (\bigcup_{j < ka}. RF\ j\ x) = (\bigcup_{j < ka}. RF\ j\ y)$   
**by** (*simp add: sorted-list-of-set-inject*)  
**show**  $x = y$   
**proof** –  
**obtain**  $n$  **where**  $n: n \in RF\ 0\ x$   
**using** *AF-ne QF-0*  $\langle 0 < k \rangle$  *Inf-nat-def1*  $\langle k \leq ka \rangle$  **by** (*force simp: RF-def*)  
**with**  $eq\ \langle ka > 0 \rangle$  **obtain**  $j'$  **where**  $j' < ka\ n \in RF\ j'\ y$   
**by** *blast*  
**then show** *?thesis*  
**using** *disjoint-QF*  $[of\ k\ 0\ x\ j']\ n\ \langle x < m \rangle\ \langle y < m \rangle\ \langle ka \leq Suc\ k \rangle\ \langle 0 < k \rangle$   
**by** (*force simp: RF-def disjnt-iff simp del: QF-0 split: if-split-asm*)  
**qed**  
**qed**

**define**  $M$  **where**  $M \equiv (\lambda i. \text{list-of } (\bigcup_{j < ka}. RF\ j\ i))\ \text{'}\{..<m\}$   
**have** *finite*  $M$   
**unfolding** *M-def* **by** *blast*  
**moreover have**  $card\ M = m$   
**by** (*simp add: M-def*  $\langle k \leq ka \rangle$  *card-image inj*)  
**moreover have**  $M \subseteq WW$   
**by** (*force simp: M-def WW-def*)  
**ultimately show**  $M \in [WW]^m$   
**by** (*simp add: nsets-def*)

**have** *sm-RF: strict-mono-sets*  $\{..<ka\}$   $(\lambda j. RF\ j\ i)$  **if**  $i < m$  **for**  $i$   
**using** *sm-QF1* *that kka*  
**by** (*simp add: less-QF RF-def strict-mono-sets-def*)

**have** *RF-non-Nil: list-of*  $(RF\ j\ i) \neq []$  **if**  $j < Suc\ k$  **for**  $i\ j$   
**using** *that* **by** (*simp add: RF-def*)

**have** *less-RF-same: RF*  $j\ i' \ll RF\ j\ i$   
**if**  $i' < i\ j < k$  **for**  $i'\ i\ j$   
**using** *that* **by** (*simp add: less-QF-same RF-def*)

**have** *less-RF-same-k: RF*  $k\ i' \ll RF\ k\ i$  — reversed version for  $k$   
**if**  $i < i'\ i' < m$  **for**  $i'\ i$   
**using** *that* **by** (*simp add: less-QF-same RF-def*)

**show** *Form*  $l\ U \wedge \text{list.set } (inter\ scheme\ l\ U) \subseteq N$  **if**  $U \in [M]^2$  **for**  $U$

**proof** –  
**from** *that* **obtain**  $x\ y$  **where**  $U = \{x, y\}\ x \in M\ y \in M$  **and**  $xy: (x, y) \in \text{lenlex}$   
*less-than*  
**by** (*auto simp: lenlex-nsets-2-eq*)  
**let**  $?R = \lambda p. \text{list-of } \circ (\lambda j. RF\ j\ p)$   
**obtain**  $p\ q$  **where**  $x: x = \text{list-of } (\bigcup_{j < ka}. RF\ j\ p)$   
**and**  $y: y = \text{list-of } (\bigcup_{j < ka}. RF\ j\ q)$  **and**  $p < m\ q < m$

```

    using ⟨x ∈ M⟩ ⟨y ∈ M⟩ by (auto simp: M-def)
  then have pq: p < q length x < length y
    using xy ⟨k ≤ ka⟩ ⟨ka ≤ Suc k⟩ le1-not-refl [OF irrefl-less-than]
    by (auto simp: lenlex-def sm-RF sorted-list-of-set-UN-lessThan length-concat
sum-sorted-list-of-set-map)
  have xc: x = concat (map (?R p) (list-of {..<ka}))
    by (simp add: x sorted-list-of-set-UN-eq-concat ⟨k ≤ ka⟩ ⟨ka ≤ Suc k⟩ ⟨p <
m⟩ sm-RF)
  have yc: y = concat (map (?R q) (list-of {..<ka}))
    by (simp add: y sorted-list-of-set-UN-eq-concat ⟨k ≤ ka⟩ ⟨ka ≤ Suc k⟩ ⟨q <
m⟩ sm-RF)
  have enum-DF-AF: enum (DF k p) (ka - 1) < hd (list-of (AF k p)) for p
  proof (rule less-setsD [OF DF-AF])
    show enum (DF k p) (ka - 1) ∈ DF k p
    using ⟨ka ≤ Suc k⟩ card-DF finite-DF by (auto simp: finite-enumerate-in-set)
    show hd (list-of (AF k p)) ∈ AF k p
      using AF-non-Nil finite-AF hd-in-set set-sorted-list-of-set by blast
  qed

  have less-RF-RF: RF n p ≪ RF n q if n < k for n
    using that ⟨p < q⟩ by (simp add: less-RF-same)
  have less-RF-Suc: RF n q ≪ RF (Suc n) q if n < k for n
    using ⟨q < m⟩ that by (auto simp: RF-def less-QF)
  have less-RF-k: RF k q ≪ RF k p
    using ⟨q < m⟩ less-RF-same-k ⟨p < q⟩ by blast
  have less-RF-k-ka: RF (k-1) p ≪ RF (ka - 1) q
    using ka-k-or-Suc less-RF-RF
    by (metis One-nat-def RF-def ⟨0 < k⟩ ⟨ka - 1 ≤ k⟩ ⟨p < m⟩ diff-Suc-1
diff-Suc-less less-QF-step)
  have Inf-DF-eq-enum: ⌊ (DF k i) = enum (DF k i) 0 for k i
    by (simp add: Inf-nat-def enumerate-0)

  have Inf-DF-less: ⌊ (DF k i') < ⌊ (DF k i) if i' < i for i' i k
    by (metis DF-ne enum-0-eq-Inf enum-0-eq-Inf-finite enum-DF-less-iff le0
that)
  have AF-Inf-DF-less: ⌋ x. x ∈ AF k i ⇒ ⌊ (DF k i') < x if i' ≤ i for i' i k
    using less-setsD [OF DF-AF] DF-ne that
  by (metis Inf-DF-less Inf-nat-def1 dual-order.order-iff-strict dual-order.strict-trans)

  show ?thesis — The general case requires 1 < k, necessitating a painful special
case
  proof (cases k=1)
    case True
      with kka consider ka=1 | ka=2 by linarith
      then show ?thesis
    proof cases
      case 1
        define zs where zs = card (AF 1 p) # list-of (AF 1 p)
          @ card (AF 1 q) # list-of (AF 1 q)

```



```

have zs: Form-Body ka k x y zs
proof (intro that exI conjI Form-Body.intros)
  show x = concat ([list-of (AF k p)]) y = concat ([list-of (AF k q)])
    by (simp-all add: x y 1 lessThan-Suc RF-0)
  have AF k p  $\ll$  insert ( $\sqcap$  (DF k q)) (AF k q)
  by (metis AF-DF DF-ne Inf-nat-def1 RF-0  $\langle 0 < k \rangle$  insert-iff less-RF-RF
less-sets-def pq(1))
  then have strict-sorted (list-of (AF k p) @  $\sqcap$  (DF k q) # list-of (AF k
q))
    by (auto simp: strict-sorted-append-iff intro: less-sets-imp-list-less
AF-Inf-DF-less)
  moreover have  $\bigwedge x. x \in AF\ k\ q \implies \sqcap (DF\ k\ p) < x$ 
    by (meson AF-Inf-DF-less less-imp-le-nat  $\langle p < q \rangle$ )
  moreover have  $\bigwedge x. x \in AF\ 1\ p \implies \sqcap (DF\ 1\ p) < x$ 
    by (meson DF-AF DF-ne Inf-nat-def1 less-setsD)
  ultimately show strict-sorted zs
    using  $\langle p < q \rangle$  True Inf-DF-less DF-AF DF-ne
    by (auto simp: zs-def less-sets-def card-AF AF-Inf-DF-less)
qed (auto simp:  $\langle k=1 \rangle$   $\langle ka=1 \rangle$  zs-def AF-ne  $\langle \text{length } x < \text{length } y \rangle$ )
have zs-N: list.set zs  $\subseteq$  N
  using AF-subset-N by (auto simp: zs-def card-AF Inf-DF-N  $\langle k=1 \rangle$ )
show ?thesis
proof
  have l = 1
    using kka  $\langle k=1 \rangle$   $\langle ka=1 \rangle$  by auto
  have Form (2*1-1) {x,y}
    using 1 Form.intros(2) True zs by fastforce
  then show Form l U
    by (simp add:  $\langle U = \{x,y\} \rangle$   $\langle l = 1 \rangle$  One-nat-def)
  show list.set (inter-scheme l U)  $\subseteq$  N
    using kka zs zs-N  $\langle k=1 \rangle$  Form-Body-imp-inter-scheme by (fastforce
simp:  $\langle U = \{x,y\} \rangle$ )
  qed
next
  case 2 — Still in our painful special case
  note True [simp] note 2 [simp]
  have [simp]:  $\{0 < .. < 2\} = \{1 :: nat\}$ 
    by auto

  have enum-DF1-eq: enum (DF 1 i) 1 = card (AF 1 i) + card (RF 1 i)
    if i < m for i
    using card-AF-sum that by (simp add: One-nat-def)
  have card-RF: card (RF 1 i) = enum (DF 1 i) 1 - enum (DF 1 i) 0 if i
< m for i
    using that by (auto simp: RF-def card-QF del-def)
  have list-of-AF-RF: list-of (AF 1 q  $\cup$  RF 1 q) = list-of (AF 1 q) @ list-of
(RF 1 q)
    by (metis One-nat-def RF-0 True  $\langle 0 < k \rangle$  finite-RF less-RF-Suc
sorted-list-of-set-Un)

```

```

define zs where zs = card (AF 1 p) # (card (AF 1 p) + card (RF 1 p))
# list-of (AF 1 p)
      @ (card (AF 1 q) + card (RF 1 q)) # list-of (AF 1 q) @ list-of (RF
1 q) @ list-of (RF 1 p)
have zs: Form-Body ka k x y zs
proof (intro that exI conjI Form-Body.intros)
      have x = list-of (RF 0 p ∪ RF 1 p)
      by (simp add: x eval-nat-numeral lessThan-Suc RF-0 Un-commute
One-nat-def)
      also have ... = list-of (RF 0 p) @ list-of (RF 1 p)
      using RF-def True ⟨p < m⟩ less-QF-step
      by (metis QF-0 RF-0 diff-self-eq-0 finite-RF le-refl sorted-list-of-set-Un
zero-less-one)
      finally show x = concat ([list-of (AF 1 p),list-of (RF 1 p)])
      by (simp add: RF-0)

      have *: i ∈ AF 1 q ∨ i ∈ RF 1 q ∨ i ∈ RF 1 p ⇒ enum (DF 1 q) 1
< i for i
      using True card-DF finite-enumerate-in-set[OF finite-DF]
      by (metis AF-ne DF-AF One-nat-def RF-0 RF-non-Nil finite-RF lessI
less-RF-Suc less-RF-k less-setsD
      less-sets-trans sorted-list-of-set.sorted-key-list-of-set-eq-Nil-iff)

      show y = concat [list-of (RF 1 q ∪ AF 1 q)]
      by (simp add: y eval-nat-numeral lessThan-Suc RF-0 One-nat-def)
      show zs: zs = concat [[card (AF 1 p), card (AF 1 p) + card (RF 1 p)],
list-of (AF 1 p),
      [ card (AF 1 q) + card (RF 1 q), list-of (RF 1 q ∪ AF
1 q) ] @ interact [list-of (RF 1 p) ] []
      using list-of-AF-RF by (simp add: zs-def Un-commute)

      show strict-sorted zs
      proof (simp add: ⟨p < m⟩ ⟨q < m⟩ ⟨p < q⟩ zs-def strict-sorted-append-iff,
intro conjI strip)
      show 0 < card (RF 1 p)
      using ⟨p < m⟩ by (simp add: card-RF card-DF finite-DF)
      show G1: card (AF 1 p) < card (AF 1 q) + card (RF 1 q)
      by (simp add: Inf-DF-less card-AF ⟨p < q⟩ trans-less-add1)
      show card (AF 1 p) < x
      if x ∈ AF 1 p ∪ (AF 1 q ∪ (RF 1 q ∪ RF 1 p)) for x
      using that ⟨q < m⟩ *
      by (metis (no-types) order-refl AF-Inf-DF-less Un-iff G1 card-AF
order.strict-trans enum-DF1-eq that)
      show G2: card (AF 1 p) + card (RF 1 p) < card (AF 1 q) + card (RF
1 q)
      using ⟨p < q⟩ ⟨p < m⟩ ⟨q < m⟩ by (metis enum-DF1-eq enum-DF-less-iff
le-refl)
      show card (AF 1 q) + card (RF 1 q) < x

```

```

    if  $x \in AF\ 1\ q \cup (RF\ 1\ q \cup RF\ 1\ p)$  for  $x$ 
    using that  $\langle q < m \rangle * enum\text{-}DF1\text{-}eq$  by force
    then show  $card\ (AF\ 1\ p) + card\ (RF\ 1\ p) < x$ 
    if  $x \in AF\ 1\ p \cup (AF\ 1\ q \cup (RF\ 1\ q \cup RF\ 1\ p))$  for  $x$ 
    using that  $\langle p < m \rangle finite\text{-}enumerate\text{-}in\text{-}set[OF\ finite\text{-}DF]$ 
    by (metis DF-AF G2 Un-iff card-DF dual-order.strict-trans enum-DF1-eq
lessI less-setsD)
    have  $list\text{-}of\ (AF\ 1\ p) < list\text{-}of\ \{enum\ (DF\ 1\ q)\ 1\}$ 
    proof (rule less-sets-imp-sorted-list-of-set)
    show  $AF\ 1\ p \ll \{enum\ (DF\ 1\ q)\ 1\}$ 
    by (metis AF-DF card-DF empty-subsetI finite-DF finite-enumerate-in-set
insert-subset
less-Suc-eq less-sets-weaken2  $\langle p < q \rangle$ )
qed auto
    then show  $list\text{-}of\ (AF\ 1\ p) < (card\ (AF\ 1\ q) + card\ (RF\ 1\ q)) \#$ 
 $list\text{-}of\ (AF\ 1\ q) @ list\text{-}of\ (RF\ 1\ q) @ list\text{-}of\ (RF\ 1\ p)$ 
    using  $\langle q < m \rangle$  by (simp add: less-list-def enum-DF1-eq)
    have  $list\text{-}of\ (AF\ 1\ q) < list\text{-}of\ (RF\ 1\ q)$ 
    by (metis One-nat-def RF-0 True  $\langle 0 < k \rangle finite-RF less-RF-Suc$ 
less-sets-imp-sorted-list-of-set)
    then show  $list\text{-}of\ (AF\ 1\ q) < list\text{-}of\ (RF\ 1\ q) @ list\text{-}of\ (RF\ 1\ p)$ 
    using RF-non-Nil by (auto simp: less-list-def)
    show  $list\text{-}of\ (RF\ 1\ q) < list\text{-}of\ (RF\ 1\ p)$ 
    using True finite-RF less-RF-k less-sets-imp-sorted-list-of-set by metis
qed
    show  $[list\text{-}of\ (AF\ 1\ p), list\text{-}of\ (RF\ 1\ p)] \in lists\ (-\ \{\ \})$ 
    using RF-non-Nil  $\langle 0 < k \rangle$  by (auto simp: zs-def AF-ne)
    show  $[card\ (AF\ 1\ q) + card\ (RF\ 1\ q)] = acc\text{-}lengths\ 0\ [list\text{-}of\ (RF\ 1\ q$ 
 $\cup\ AF\ 1\ q)]$ 
    using list-of-AF-RF by (auto simp: zs-def AF-ne sup-commute)
qed (auto simp: zs-def AF-ne  $\langle length\ x < length\ y \rangle$ )
have  $zs\text{-}N: list.set\ zs \subseteq N$ 
    using  $\langle p < m \rangle \langle q < m \rangle DF\text{-}in\text{-}N\ enum\text{-}DF1\text{-}eq$  [symmetric]
    by (auto simp: zs-def card-AF AF-subset-N RF-subset-N Inf-DF-N)
show ?thesis
proof
    have Form (2*1)  $\{x,y\}$ 
    by (metis 2 Form.simps Suc-1 True zero-less-one zs)
    with kka show Form l U
    by (simp add:  $\langle U = \{x,y\} \rangle$ )
    show  $list.set\ (inter\text{-}scheme\ l\ U) \subseteq N$ 
    using kka zs zs-N  $\langle k=1 \rangle Form\text{-}Body\text{-}imp\text{-}inter\text{-}scheme$  by (fastforce
simp:  $\langle U = \{x, y\} \rangle$ )
    qed
    qed
next
case False
then have  $k \geq 2\ ka \geq 2$ 
    using kka  $\langle k > 0 \rangle$  by auto

```

**then have** *k-minus-1* [*simp*]:  $Suc (k - Suc (Suc 0)) = k - Suc 0$   
**by** *auto*  
**have** [*simp*]:  $Suc (k - 2) = k - 1$   
**using**  $\langle k \geq 2 \rangle$  **by** *linarith*  
**define** *PP* **where**  $PP \equiv map (?R p) (list-of \{0 <..<ka\})$   
**define** *QQ* **where**  $QQ \equiv map (?R q) (list-of \{0 <..<k-1\}) @ ([list-of (RF$   
 $(k-1) q \cup RF (ka-1) q])$   
**let** *?INT* = *interact PP QQ*  
— No separate sets A and B as in the text, but instead we treat both cases  
as once  
**have** [*simp*]:  $length PP = ka - 1$   
**by** (*simp add: PP-def*)  
**have** [*simp*]:  $length QQ = k-1$   
**using**  $\langle k \geq 2 \rangle$  **by** (*simp add: QQ-def*)  
  
**have** *PP-n*:  $PP ! n = list-of (RF (Suc n) p)$   
**if**  $n < ka-1$  **for**  $n$   
**using** *that kka* **by** (*auto simp: PP-def nth-sorted-list-of-set-greaterThanLessThan*)  
  
**have** *QQ-n*:  $QQ ! n = (if n < k-2 then list-of (RF (Suc n) q)$   
 $else list-of (RF (k-1) q \cup RF (ka - 1) q))$   
**if**  $n < k-1$  **for**  $n$   
**using** *that kka* **by** (*auto simp: QQ-def nth-append nth-sorted-list-of-set-greaterThanLessThan*)  
  
**have** *QQ-n-same*:  $QQ ! n = list-of (RF (Suc n) q)$   
**if**  $n < k-1$   $k=ka$  **for**  $n$   
**using** *that kka Suc-diff-Suc*  
**by** (*fastforce simp: One-nat-def QQ-def nth-append nth-sorted-list-of-set-greaterThanLessThan*)  
  
**have** *split-nat-interval*:  $\{0 <..<n\} = insert (n-1) \{0 <..<n-1\}$  **if**  $n \geq 2$   
**for**  $n::nat$   
**using** *that* **by** *auto*  
**have** *split-list-interval*:  $list-of\{0 <..<n\} = list-of\{0 <..<n-1\} @ [n-1]$  **if**  $n$   
 $\geq 2$  **for**  $n::nat$   
**proof** (*intro sorted-list-of-set-unique [THEN iffD1] conjI*)  
**have**  $list-of \{0 <..<n - 1\} < [n - 1]$   
**by** (*auto intro: less-sets-imp-list-less*)  
**then show** *strict-sorted* ( $list-of \{0 <..<n - 1\} @ [n - 1]$ )  
**by** (*auto simp: strict-sorted-append-iff*)  
**qed** (*use*  $\langle n \geq 2 \rangle$  **in** *auto*)  
  
**have** *list-of-RF-Un*:  $list-of (RF (k-1) q \cup RF k q) = list-of (RF (k-1) q)$   
 $@ list-of (RF k q)$   
**by** (*metis Suc-diff-1*  $\langle 0 < k \rangle$  *finite-RF lessI less-RF-Suc sorted-list-of-set-Un*)  
  
**have** *card-AF-sum-QQ*:  $card (AF k q) + sum-list (map length QQ) =$   
 $(\sum j < ka. card (RF j q))$   
**proof** (*cases ka = Suc k*)  
**case** *True*

```

have  $RF\ (k-1)\ q \cap RF\ k\ q = \{\}$ 
  using less-RF-Suc [of k-1] <k > 0> by (auto simp: less-sets-def)
  then have  $card\ (RF\ (k-1)\ q \cup RF\ k\ q) = card\ (RF\ (k-1)\ q) + card$ 
( $RF\ k\ q$ )
  by (simp add: card-Un-disjoint)
then show ?thesis
  using  $\langle k \geq 2 \rangle\ \langle q < m \rangle$ 
  apply (simp add: QQ-def True flip: RF-0)
apply (simp add: lessThan-k split-nat-interval sum-sorted-list-of-set-map)
done
next
  case False
  with  $kka$  have  $ka=k$  by linarith
with  $\langle k \geq 2 \rangle$  show ?thesis by (simp add: QQ-def lessThan-k split-nat-interval
sum-sorted-list-of-set-map flip: RF-0)
qed

define LENS where  $LENS \equiv \lambda i. acc-lengths\ 0\ (list-of\ (AF\ k\ i)\ \# \ map$ 
( $?R\ i$ ) ( $list-of\ \{0 < .. < ka\}$ ))
have LENS-subset-N:  $list.set\ (LENS\ i) \subseteq N$  if  $i < m$  for  $i$ 
proof -
  have eq:  $(list-of\ (AF\ k\ i)\ \# \ map\ (?R\ i)\ (list-of\ \{0 < .. < ka\})) = map\ (?R$ 
 $i)$  ( $list-of\ \{.. < ka\}$ )
  using RF-0 <0 < ka> sorted-list-of-set-k by auto
  let  $?f = rec-nat\ [card\ (AF\ k\ i)]\ (\lambda n\ r. r\ @\ [(\sum\ j \leq Suc\ n. card\ (RF\ j\ i))])$ 
  have  $f$ :  $acc-lengths\ 0\ (map\ (?R\ i)\ (list-of\ \{..v\})) = ?f\ v$  for  $v$ 
by (induction v) (auto simp: RF-0 acc-lengths-append sum-sorted-list-of-set-map)
  have  $\exists$ :  $list.set\ (?f\ v) \subseteq N$  if  $v \leq k$  for  $v$ 
  using that
proof (induction v)
  case  $0$ 
  have  $card\ (AF\ k\ i) \in N$ 
  by (metis DF-N DF-ne Inf-nat-def1 card-AF subsetD)
  with  $0$  show ?case by simp
next
  case (Suc v)
  then have  $enum\ (DF\ k\ i)\ (Suc\ v) \in N$ 
  by (metis DF-N card-DF finite-enumerate-in-set finite-DF in-mono
le-imp-less-Suc)
  with  $Suc\ \langle i < m \rangle$  show ?case
  by (simp del: sum.atMost-Suc)
qed
show ?thesis
  unfolding LENS-def
  by (metis \exists Suc-pred' <0 < ka> <ka - 1 \leq k> eq f lessThan-Suc-atMost)
qed
define LENS-QQ where  $LENS-QQ \equiv acc-lengths\ 0\ (list-of\ (AF\ k\ q)\ \#$ 
 $QQ)$ 
have LENS-QQ-subset:  $list.set\ LENS-QQ \subseteq list.set\ (LENS\ q)$ 

```

```

proof (cases ka = Suc k)
  case True
    with ⟨k ≥ 2⟩ show ?thesis
      unfolding QQ-def LENS-QQ-def LENS-def
      by (auto simp: list-of-RF-Un split-list-interval acc-lengths-append)
  next
    case False
      then have ka=k
        using kka by linarith
      with ⟨k ≥ 2⟩ show ?thesis
        by (simp add: QQ-def LENS-QQ-def LENS-def split-list-interval)
  qed
have ss-INT: strict-sorted ?INT
proof (rule strict-sorted-interact-I)
  fix n
  assume n < length QQ
  then have n: n < k-1
    by simp
  have n = k - 2 if ¬ n < k - 2
    using n that by linarith
  moreover have list-of (RF (Suc (k - 2)) p) < list-of (RF (k-1) q) ∪
RF (ka - 1) q)
    by (auto simp: less-sets-imp-sorted-list-of-set less-sets-Un2 less-RF-RF
less-RF-k-ka ⟨0 < k⟩)
  ultimately show PP ! n < QQ ! n
  using ⟨k ≤ ka⟩ n by (auto simp: PP-n QQ-n less-sets-imp-sorted-list-of-set
less-RF-RF)
  next
    fix n
    have V: [Suc n < ka - 1] ⇒ list-of (RF (Suc n) q) < list-of (RF (Suc
(Suc n)) p) for n
      by (smt RF-def Suc-leI ⟨ka - 1 ≤ k⟩ ⟨q < m⟩ diff-Suc-1 finite-RF
less-QF-step less-le-trans less-sets-imp-sorted-list-of-set nat-neq-iff zero-less-Suc)
    have RF (k - 1) q ≪ RF k p
      by (metis One-nat-def RF-non-Nil Suc-pred ⟨0 < k⟩ finite-RF lessI
less-RF-Suc less-RF-k less-sets-trans sorted-list-of-set-eq-Nil-iff)
    with kka have RF (k-1) q ∪ RF (ka - 1) q ≪ RF k p
      by (metis less-RF-k One-nat-def less-sets-Un1 antisym-conv2 diff-Suc-1
le-less-Suc-eq)
    then have VI: list-of (RF (k-1) q) ∪ RF (ka - 1) q < list-of (RF k p)
      by (rule less-sets-imp-sorted-list-of-set) auto
    assume Suc n < length PP
    with ⟨ka ≤ Suc k⟩ VI
    show QQ ! n < PP ! Suc n
      apply (clarsimp simp: PP-n QQ-n V)
      by (metis One-nat-def Suc-1 Suc-lessI add.right-neutral add-Suc-right
diff-Suc-Suc ka-k-or-Suc less-diff-conv)
  next
    show PP ∈ lists (- {[]})

```

```

    using RF-non-Nil kka
    by (clarsimp simp: PP-def) (metis RF-non-Nil less-le-trans)
  show QQ ∈ lists (- {})
    using RF-non-Nil kka
    by (clarsimp simp: QQ-def) (metis RF-non-Nil Suc-pred ⟨0 < k⟩ less-SucI
One-nat-def)
  qed (use kka PP-def QQ-def in auto)
  then have ss-QQ: strict-sorted (concat QQ)
    using strict-sorted-interact-imp-concat by blast

obtain zs where zs: Form-Body ka k x y zs and zs-N: list.set zs ⊆ N
proof (intro that exI conjI Form-Body.intros [OF ⟨length x < length y⟩])
  show x = concat (list-of (AF k p) # PP)
    using ⟨ka > 0⟩ by (simp add: PP-def RF-0 xc sorted-list-of-set-k)
  let ?YR = (map (list-of ∘ (λj. RF j q)) (list-of {0<..

```

**using** *RF-non-Nil kka* **by** (*auto simp: AF-ne PP-def QQ-def eq-commute*  
*[of []]*)  
**show** *list.set ((LENS p @ list-of (AF k p) @ LENS-QQ @ list-of (AF k*  
*q) @ ?INT))*  $\subseteq N$   
**using** *AF-subset-N RF-subset-N LENS-subset-N*  $\langle p < m \rangle \langle q < m \rangle$   
*LENS-QQ-subset*  
**by** (*auto simp: subset-iff PP-def QQ-def*)  
**show** *length (list-of (AF k p) # PP) = ka length (list-of (AF k q) # QQ)*  
 $= k$   
**using**  $\langle 0 < ka \rangle \langle 0 < k \rangle$  **by** *auto*  
**show** *LENS p = acc-lengths 0 (list-of (AF k p) # PP)*  
**by** (*auto simp: LENS-def PP-def*)  
**show** *strict-sorted (LENS p @ list-of (AF k p) @ LENS-QQ @ list-of (AF*  
*k q) @ ?INT)*  
**unfolding** *strict-sorted-append-iff*  
**proof** (*intro conjI ss-INT*)  
**show** *LENS p < list-of (AF k p) @ LENS-QQ @ list-of (AF k q) @*  
*?INT*  
**using** *AF-non-Nil [of k p]*  $\langle k \leq ka \rangle \langle ka \leq \text{Suc } k \rangle \langle p < m \rangle$  *card-AF-sum*  
*enum-DF-AF*  
**by** (*simp add: enum-DF-AF less-list-def card-AF-sum LENS-def*  
*sum-sorted-list-of-set-map*  
*del: acc-lengths.simps*)  
**show** *strict-sorted (LENS p)*  
**unfolding** *LENS-def*  
**by** (*rule strict-sorted-acc-lengths*) (*use RF-non-Nil AF-non-Nil kka in*  
 *$\langle auto simp: in-lists-conv-set \rangle$* )  
**show** *strict-sorted LENS-QQ*  
**unfolding** *LENS-QQ-def QQ-def*  
**by** (*rule strict-sorted-acc-lengths*) (*use RF-non-Nil AF-non-Nil kka in*  
 *$\langle auto simp: in-lists-conv-set \rangle$* )  
**have** *last-AF-DF: last (list-of (AF k p)) < [] (DF k q)*  
**using** *AF-DF [OF*  $\langle p < q \rangle$ , *of k] AF-non-Nil [of k p] DF-ne [of k q]*  
**by** (*metis Inf-nat-def1 finite-AF last-in-set less-sets-def set-sorted-list-of-set*)  
**then show** *list-of (AF k p) < LENS-QQ @ list-of (AF k q) @ ?INT*  
**by** (*simp add: less-list-def card-AF LENS-QQ-def*)  
**show** *LENS-QQ < list-of (AF k q) @ ?INT*  
**using** *AF-non-Nil [of k q]*  $\langle q < m \rangle$  *card-AF-sum enum-DF-AF*  
*card-AF-sum-QQ*  
**by** (*auto simp: less-list-def AF-ne hd-append card-AF-sum LENS-QQ-def*)  
**show** *list-of (AF k q) < ?INT*  
**proof** –  
**have** *AF k q*  $\ll$  *RF 1 p*  
**using**  $\langle 0 < k \rangle \langle p < m \rangle \langle q < m \rangle$  **by** (*simp add: RF-def less-QF flip:*  
*QF-0*)  
**then have** *last (list-of (AF k q)) < hd (list-of (RF 1 p))*  
**proof** (*rule less-setsD*)  
**show** *last (list-of (AF k q))*  $\in$  *AF k q*  
**using** *AF-non-Nil finite-AF last-in-set set-sorted-list-of-set* **by** *blast*



```

      show hd (list-of (RF 1 p)) ∈ RF 1 p
      by (metis One-nat-def RF-non-Nil ⟨0 < k⟩ finite-RF hd-in-set
not-less-eq set-sorted-list-of-set)
    qed
    with ⟨k > 0⟩ ⟨ka ≥ 2⟩ RF-non-Nil show ?thesis
    by (simp add: One-nat-def hd-interact less-list-def sorted-list-of-set-greaterThanLessThan
PP-def QQ-def)
  qed
  qed auto
  qed (auto simp: LENS-QQ-def)
  show ?thesis
  proof (cases ka = k)
    case True
    then have l = 2*k-1
      by (simp add: kka(3) mult-2)
    then show ?thesis
      by (metis One-nat-def Form.intros(2) Form-Body-imp-inter-scheme True
⟨0 < k⟩ ⟨U = {x, y}⟩ kka zs zs-N)
  next
    case False
    then have l = 2*k
      using kka by linarith
    then show ?thesis
      by (metis One-nat-def False Form.intros(3) Form-Body-imp-inter-scheme
⟨0 < k⟩ ⟨U = {x, y}⟩ antisym kka le-SucE zs zs-N)
  qed
  qed
  qed
  qed
  qed

```

### 3.11 Larson's Lemma 3.8

#### 3.11.1 Primitives needed for the inductive construction of $b$

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

**lemma**  $IJ\text{-iff}$ :  $u \in IJ\ k \longleftrightarrow (\exists j\ i. u = (j, i) \wedge i < j \wedge j \leq k)$

by (auto simp:  $IJ\text{-def}$ )

**lemma**  $\text{finite-}IJ$ :  $\text{finite } (IJ\ k)$

by (auto simp:  $IJ\text{-def}$ )

**fun**  $\text{prev}$  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**  $\text{prev-eq-None-iff}$ :  $\text{prev } j\ i = \text{None} \longleftrightarrow j \leq \text{Suc } 0 \wedge i = 0$

by (auto simp: le-Suc-eq elim: prev.elims)

**lemma** *prev-pair-less*:

$prev\ j\ i = Some\ ji' \implies (ji', (j,i)) \in pair-less$

by (auto simp: pair-lessI1 elim: prev.elims)

**lemma** *prev-Some-less*:  $\llbracket prev\ j\ i = Some\ (j',i'); i \leq j \rrbracket \implies i' < j'$

by (auto elim: prev.elims)

**lemma** *prev-maximal*:

$\llbracket prev\ j\ i = Some\ (j',i'); (ji'', (j,i)) \in pair-less; ji'' \in IJ\ k \rrbracket$

$\implies (ji'', (j',i')) \in pair-less \vee ji'' = (j',i')$

by (force simp: IJ-def pair-less-def elim: prev.elims)

**lemma** *pair-less-prev*:

assumes  $(u, (j,i)) \in pair-less\ u \in IJ\ k$

shows  $prev\ j\ i = Some\ u \vee (\exists x. (u, x) \in pair-less \wedge prev\ j\ i = Some\ x)$

using *assms*

**proof** (cases *prev j i*)

case *None*

then show ?thesis

using *assms* by (force simp: prev-eq-None-iff pair-less-def IJ-def split: prod.split)

**next**

case (*Some a*)

then show ?thesis

by (metis *assms* prev-maximal prod.exhaust-sel)

**qed**

### 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  $USigma\ A\ B \equiv \bigcup X \in A. \bigcup y \in B\ X. \{insert\ y\ X\}$

**definition** *usplit*

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

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

by (auto simp: *USigma-def*)

**lemma** *USigma-iff*:

assumes  $\bigwedge I\ j. I \in \mathcal{I} \implies I \ll J\ I \wedge finite\ I$

shows  $x \in USigma\ \mathcal{I}\ J \iff usplit\ (\lambda I\ j. I \in \mathcal{I} \wedge j \in J\ I \wedge x = insert\ j\ I)\ x$

**proof** –

have [*simp*]:  $\bigwedge I\ j. \llbracket I \in \mathcal{I}; j \in J\ I \rrbracket \implies Max\ (insert\ j\ I) = j$

by (meson *Max-insert2* *assms* less-imp-le less-sets-def)

show ?thesis

**proof** –

have  $\S: j \notin I$  if  $I \in \mathcal{I}\ j \in J\ I$  for  $I\ j$

using *that* by (metis *assms* less-irrefl less-sets-def)

**have**  $\exists I \in \mathcal{I}. \exists j \in J I. x = \text{insert } j I$   
**if**  $x - \{\text{Max } x\} \in \mathcal{I}$  **and**  $\text{Max } x \in J (x - \{\text{Max } x\}) x \neq \{\}$   
**using** *that by* (*metis Max-in assms infinite-remove insert-Diff*)  
**then show** *?thesis*  
**by** (*auto simp: USigma-def usplit-def §*)  
**qed**  
**qed**

**proposition** *ordertype-append-image-IJ:*

**assumes** *lenB [simp]:*  $\bigwedge i j. i \in \mathcal{I} \implies j \in J i \implies \text{length } (B j) = c$   
**and** *AB:*  $\bigwedge i j. i \in \mathcal{I} \implies j \in J i \implies A i < B j$   
**and** *IJ:*  $\bigwedge i. i \in \mathcal{I} \implies i \ll J i \wedge \text{finite } i$   
**and**  $\beta: \bigwedge i. i \in \mathcal{I} \implies \text{ordertype } (B \text{ ' } J i) (\text{lenlex less-than}) = \beta$   
**and** *A:* *inj-on*  $A \mathcal{I}$   
**shows**  $\text{ordertype } (\text{usplit } (\lambda i j. A i @ B j) \text{ ' } \text{USigma } \mathcal{I} J) (\text{lenlex less-than})$   
 $= \beta * \text{ordertype } (A \text{ ' } \mathcal{I}) (\text{lenlex less-than})$   
*(is ordertype ?AB ?R = - \* ?α)*

**proof** (*cases*  $\mathcal{I} = \{\}$ )

**case** *False*

**have** *Ord*  $\beta$

**using**  $\beta$  *False wf-Ord-ordertype by fastforce*

**show** *?thesis*

**proof** (*subst ordertype-eq-iff*)

**define** *split where*  $\text{split} \equiv \lambda l::\text{nat list}. (\text{take } (\text{length } l - c) l, (\text{drop } (\text{length } l - c) l))$

**have** *oB:*  $\text{ordermap } (B \text{ ' } J i) ?R (B j) \sqsubset \beta$  **if**  $\langle i \in \mathcal{I} \rangle \langle j \in J i \rangle$  **for**  $i j$

**using**  $\beta$  *less-TC-iff that by fastforce*

**then show** *Ord*  $(\beta * ?\alpha)$

**by** (*intro*  $\langle \text{Ord } \beta \rangle$  *wf-Ord-ordertype Ord-mult; simp*)

**define** *f where*  $f \equiv \lambda u. \text{let } (x, y) = \text{split } u \text{ in let } i = \text{inv-into } \mathcal{I} A x \text{ in}$

$\beta * \text{ordermap } (A \text{ ' } \mathcal{I}) ?R x + \text{ordermap } (B \text{ ' } J i) ?R y$

**have** *inv-into-IA [simp]:*  $\text{inv-into } \mathcal{I} A (A i) = i$  **if**  $i \in \mathcal{I}$  **for**  $i$

**by** (*simp add: A that*)

**show**  $\exists f. \text{bij-betw } f ?AB (\text{elts } (\beta * ?\alpha)) \wedge (\forall x \in ?AB. \forall y \in ?AB. (f x < f y) = ((x, y) \in ?R))$

**unfolding** *bij-betw-def*

**proof** (*intro exI conjI strip*)

**show** *inj-on*  $f ?AB$

**proof** (*clarsimp simp: f-def inj-on-def split-def USigma-iff IJ usplit-def*)

**fix**  $x y$

**assume**  $\S: \beta * \text{ordermap } (A \text{ ' } \mathcal{I}) ?R (A (x - \{\text{Max } x\})) + \text{ordermap } (B \text{ ' } J (x - \{\text{Max } x\})) ?R (B (\text{Max } x))$

$= \beta * \text{ordermap } (A \text{ ' } \mathcal{I}) ?R (A (y - \{\text{Max } y\})) + \text{ordermap } (B \text{ ' } J$

$(y - \{\text{Max } y\})) ?R (B (\text{Max } y))$

**and**  $x: x - \{\text{Max } x\} \in \mathcal{I}$

**and**  $y: y - \{\text{Max } y\} \in \mathcal{I}$

**and**  $mx: \text{Max } x \in J (x - \{\text{Max } x\})$

**and**  $x = \text{insert } (\text{Max } x) x$

```

    and my: Max y ∈ J (y - {Max y})
    have ordermap (A'ℐ) ?R (A (x - {Max x})) = ordermap (A'ℐ) ?R (A (y
- {Max y}))
    and B-eq: ordermap (B'J (x - {Max x})) ?R (B (Max x)) = ordermap
(B'J (y - {Max y})) ?R (B (Max y))
    using mult-cancellation-lemma [OF §] oB mx my x y by blast+
    then have A (x - {Max x}) = A (y - {Max y})
    using x y by auto
    then have x - {Max x} = y - {Max y}
    by (metis x y inv-into-IA)
    then show A (x - {Max x}) = A (y - {Max y}) ∧ B (Max x) = B (Max
y)
    using B-eq mx my by auto
qed
show f' ?AB = elts (β * ?α)
proof
  show f' ?AB ⊆ elts (β * ?α)
  using ⟨Ord β⟩
  apply (clarsimp simp add: f-def split-def USigma-iff IJ usplit-def)
  by (metis TC-small β add-mult-less image-eqI ordermap-in-ordertype
trans-llt wf-Ord-ordertype wf-llt)
  show elts (β * ?α) ⊆ f' ?AB
  proof (clarsimp simp: f-def split-def image-iff USigma-iff IJ usplit-def Bex-def
elim!: elts-multE split: prod.split)
    fix γ δ
    assume δ: δ ∈ elts β and γ: γ ∈ elts ?α
    have γ ∈ ordermap (A'ℐ) (lenlex less-than) 'A'ℐ
    by (meson γ ordermap-surj subset-iff)
    then obtain i where i ∈ ℐ and yv: γ = ordermap (A'ℐ) ?R (A i)
    by blast
    have δ ∈ ordermap (B'J i) (lenlex less-than) 'B'J i
    by (metis (no-types) β δ ⟨i ∈ ℐ⟩ in-mono ordermap-surj)
    then obtain j where j ∈ J i and xu: δ = ordermap (B'J i) ?R (B j)
    by blast
    then have mji: Max (insert j i) = j
    by (meson IJ Max-insert2 ⟨i ∈ ℐ⟩ less-imp-le less-sets-def)
    have [simp]: i - {j} = i
    using IJ ⟨i ∈ ℐ⟩ ⟨j ∈ J i⟩ less-setsD by fastforce
    show ∃ l. (∃ K. K - {Max K} ∈ ℐ ∧ Max K ∈ J (K - {Max K})) ∧ K
= insert (Max K) K ∧
      l = A (K - {Max K}) @ B (Max K) ∧ β * γ + δ =
      β *
      ordermap (A'ℐ) ?R (take (length l - c) l) +
      ordermap (B'J (inv-into ℐ A (take (length l - c) l)))
      ?R (drop (length l - c) l)
  proof (intro conjI exI)
    let ?ji = insert j i
    show A i @ B j = A (?ji - {Max ?ji}) @ B (Max ?ji)
    by (auto simp: mji)
  end
end

```

```

    qed (use ⟨i ∈ I⟩ ⟨j ∈ J i⟩ mji xu yv in auto)
  qed
  qed
next
fix p q
assume p ∈ ?AB and q ∈ ?AB
then obtain x y where peq: p = A (x - {Max x}) @ B (Max x)
    and qeq: q = A (y - {Max y}) @ B (Max y)
    and x: x - {Max x} ∈ I
    and y: y - {Max y} ∈ I
    and mx: Max x ∈ J (x - {Max x})
    and my: Max y ∈ J (y - {Max y})
  by (auto simp: USigma-iff IJ usplit-def)
let ?mx = x - {Max x}
let ?my = y - {Max y}
show (f p < f q) ↔ ((p, q) ∈ ?R)
proof
  assume f p < f q
  then
  consider ordermap (A I) ?R (A (x - {Max x})) < ordermap (A I) ?R (A
(y - {Max y}))
    | ordermap (A I) ?R (A (x - {Max x})) = ordermap (A I) ?R (A (y -
{Max y}))
      ordermap (B J (x - {Max x})) ?R (B (Max x)) < ordermap (B J (y -
{Max y})) ?R (B (Max y))
    using x y mx my
  by (auto dest: mult-cancellation-less simp: f-def split-def peq qeq oB)
then have (A ?mx @ B (Max x), A ?my @ B (Max y)) ∈ ?R
proof cases
  case 1
  then have (A ?mx, A ?my) ∈ ?R
  using x y by (force simp: Ord-mem-iff-lt intro: converse-ordermap-mono)
  then show ?thesis
  using x y mx my lenB lenlex-append1 by blast
  case 2
  then have A ?mx = A ?my
  using ⟨?my ∈ I⟩ ⟨?mx ∈ I⟩ by auto
  then have eq: ?mx = ?my
  by (metis ⟨?my ∈ I⟩ ⟨?mx ∈ I⟩ inv-into-IA)
  then have (B (Max x), B (Max y)) ∈ ?R
  using mx my 2 by (force simp: Ord-mem-iff-lt intro: converse-ordermap-mono)
  with 2 show ?thesis
  by (simp add: eq irreft-less-than)
qed
then show (p, q) ∈ ?R
  by (simp add: peq qeq f-def split-def sorted-list-of-set-Un AB)
next
assume pqR: (p, q) ∈ ?R

```

```

then have §: (A ?mx @ B (Max x), A ?my @ B (Max y)) ∈ ?R
  using peq qeq by blast
then consider (A ?mx, A ?my) ∈ ?R | A ?mx = A ?my ∧ (B (Max x), B
(Max y)) ∈ ?R
  proof (cases (A ?mx, A ?my) ∈ ?R)
    case False
      have False if (A ?my, A ?mx) ∈ ?R
        by (metis ‹?my ∈ ℐ› ‹?mx ∈ ℐ› § ‹(Max y) ∈ J ?my› ‹(Max x) ∈ J
?mx› lenB lenlex-append1 omega-sum-1-less order.asym that)
      then have A ?mx = A ?my
        by (meson False UNIV-I total-llt total-on-def)
      then show ?thesis
        using § irrefl-less-than that by auto
    qed (use that in blast)
then have β * ordermap (A ℐ) ?R (A ?mx) + ordermap (B ‘ J ?mx) ?R (B
(Max x))
  < β * ordermap (A ℐ) ?R (A ?my) + ordermap (B ‘ J ?my) ?R (B
(Max y))
  proof cases
    case 1
      show ?thesis
      proof (rule add-mult-less-add-mult)
        show ordermap (A ℐ) (lenlex less-than) (A ?mx) < ordermap (A ℐ)
(lenlex less-than) (A ?my)
          by (simp add: 1 ‹?my ∈ ℐ› ‹?mx ∈ ℐ› ordermap-mono-less)
        show Ord (ordertype (A ℐ) ?R)
          using wf-Ord-ordertype by blast
        show ordermap (B ‘ J ?mx) ?R (B (Max x)) ∈ elts β
          using Ord-less-TC-mem ‹Ord β› ‹?mx ∈ ℐ› ‹(Max x) ∈ J ?mx› oB
by blast
        show ordermap (B ‘ J ?my) ?R (B (Max y)) ∈ elts β
          using Ord-less-TC-mem ‹Ord β› ‹?my ∈ ℐ› ‹(Max y) ∈ J ?my› oB
by blast
      qed (use ‹?my ∈ ℐ› ‹?mx ∈ ℐ› ‹Ord β› in auto)
    next
      case 2
        with ‹?mx ∈ ℐ› show ?thesis
          using ‹(Max y) ∈ J ?my› ‹(Max x) ∈ J ?mx› ordermap-mono-less
          by (metis (no-types, opaque-lifting) Kirby.add-less-cancel-left TC-small
image-iff inv-into-IA trans-llt wf-llt y)
        qed
        then show f p < f q
          using ‹?my ∈ ℐ› ‹?mx ∈ ℐ› ‹(Max y) ∈ J ?my› ‹(Max x) ∈ J ?mx›
          by (auto simp: peq qeq f-def split-def AB)
        qed
      qed
    qed auto
  qed auto

```

### 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]  $\Rightarrow$  bool  
**where** *NullNull*: *merge* [] [] [] []  
| *Null*: *as*  $\neq$  []  $\implies$  *merge as* [] [concat *as*] []  
| *App*: [*as1*  $\neq$  []; *bs1*  $\neq$  [];  
concat *as1* < concat *bs1*; concat *bs1* < concat *as2*; *merge as2 bs2 as*  
*bs*]  
 $\implies$  *merge (as1@as2) (bs1@bs2) (concat as1 # as) (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*  $\neq$  []; concat *bs*  $\neq$  []]  $\implies$  *merge as bs* [concat  
*as*] [concat *bs*]

**using** *merge.App* [of *as bs* [] []]

**by** (*fastforce simp: less-list-def*)

**lemma** *merge-length1-nonempty*:

**assumes** *merge as bs us vs as*  $\in$  *lists* (- {[]})

**shows** *us*  $\in$  *lists* (- {[]})

**using** *assms* **by** *induction (auto simp: mem-lists-non-Nil)*

**lemma** *merge-length2-nonempty*:

**assumes** *merge as bs us vs bs*  $\in$  *lists* (- {[]})

**shows** *vs*  $\in$  *lists* (- {[]})

**using** *assms* **by** *induction (auto simp: mem-lists-non-Nil)*

**lemma** *merge-length1-gt-0*:

**assumes** *merge as bs us vs as*  $\neq$  []

**shows** *length us* > 0

**using** *assms* **by** *induction auto*

**lemma** *merge-length-le*:

**assumes** *merge as bs us vs*

**shows** *length vs*  $\leq$  *length us*

**using** *assms* **by** *induction auto*

**lemma** *merge-length-le-Suc*:

**assumes** *merge as bs us vs*

**shows** *length us*  $\leq$  *Suc (length vs)*

**using** *assms* **by** *induction auto*

**lemma** *merge-length-less2*:

**assumes** *merge as bs us vs*

**shows** *length vs*  $\leq$  *length as*

**using** *assms*

**proof** *induction*

**case** (*App as1 bs1 as2 bs2 as bs*)

```

then show ?case
  using length-greater-0-conv [of as1] by (simp, presburger)
qed auto

lemma merge-preserves:
  assumes merge as bs us vs
  shows concat as = concat us  $\wedge$  concat bs = concat vs
  using assms by induction auto

lemma merge-interact:
  assumes merge as bs us vs strict-sorted (concat as) strict-sorted (concat bs)
    bs  $\in$  lists ( $-$  {[]})
  shows strict-sorted (interact us vs)
  using assms
proof induction
  case (App as1 bs1 as2 bs2 as bs)
  then have bs: concat bs1 < concat bs concat bs1 < concat as
    and nonmt: concat bs1  $\neq$  []
    using merge-preserves strict-sorted-append-iff by fastforce+
  then have concat bs1 < interact as bs
    unfolding less-list-def using App bs
    by (metis (no-types, lifting) Un-iff concat-append hd-in-set last-in-set merge-preserves
set-interact sorted-wrt-append strict-sorted-append-iff)
  with App show ?case
    by (metis append-in-lists-conv concat-append hd-append2 interact.simps(3)
less-list-def strict-sorted-append-iff nonmt)
qed auto

lemma acc-lengths-merge1:
  assumes merge as bs us vs
  shows list.set (acc-lengths k us)  $\subseteq$  list.set (acc-lengths k as)
  using assms
proof (induction arbitrary: k)
  case (App as1 bs1 as2 bs2 as bs)
  then show ?case
    apply (simp add: acc-lengths-append strict-sorted-append-iff length-concat-acc-lengths)
    by (simp add: le-supI2 length-concat)
qed (auto simp: length-concat-acc-lengths)

lemma acc-lengths-merge2:
  assumes merge as bs us vs
  shows list.set (acc-lengths k vs)  $\subseteq$  list.set (acc-lengths k bs)
  using assms
proof (induction arbitrary: k)
  case (App as1 bs1 as2 bs2 as bs)
  then show ?case
    apply (simp add: acc-lengths-append strict-sorted-append-iff length-concat-acc-lengths)
    by (simp add: le-supI2 length-concat)
qed (auto simp: length-concat-acc-lengths)

```



```

lemma length-hd-le-concat:
  assumes  $as \neq []$  shows  $length\ (hd\ as) \leq length\ (concat\ as)$ 
  by (metis (no-types) add.commute assms concat.simps(2) le-add2 length-append
list.exhaust-sel)

lemma length-hd-merge2:
  assumes merge as bs us vs
  shows  $length\ (hd\ bs) \leq length\ (hd\ vs)$ 
  using assms by induction (auto simp: length-hd-le-concat)

lemma merge-less-sets-hd:
  assumes merge as bs us vs strict-sorted (concat as) strict-sorted (concat bs) bs
 $\in lists\ (-\ \{\}\}$ 
  shows  $list.set\ (hd\ us) \ll list.set\ (concat\ vs)$ 
  using assms
proof induction
  case (App as1 bs1 as2 bs2 as bs)
  then have  $\S$ :  $list.set\ (concat\ bs1) \ll list.set\ (concat\ bs2)$ 
    by (force simp: dest: strict-sorted-imp-less-sets)
  have  $*$ :  $list.set\ (concat\ as1) \ll list.set\ (concat\ bs1)$ 
    using App by (metis concat-append strict-sorted-append-iff strict-sorted-imp-less-sets)
  then have  $list.set\ (concat\ as1) \ll list.set\ (concat\ bs)$ 
    using App  $\S$  less-sets-trans merge-preserves
  by (metis List.set-empty append-in-lists-conv le-zero-eq length-0-conv length-concat-ge)
  with  $*$  App.hyps show ?case
  by (fastforce simp: less-sets-UN1 less-sets-UN2 less-sets-Un2)
qed auto

lemma set-takeWhile:
  assumes strict-sorted (concat as)  $as \in lists\ (-\ \{\}\}$ 
  shows  $list.set\ (takeWhile\ (\lambda x.\ x < y)\ as) = \{x \in list.set\ as.\ x < y\}$ 
  using assms
proof (induction as)
  case (Cons a as)
  have  $a < y$ 
    if  $a < concat\ as$  strict-sorted a strict-sorted (concat as)  $x < y\ x \neq []\ x \in$ 
list.set as
    for  $x$ 
  proof –
  have  $\S$ :  $last\ x \in list.set\ (concat\ as)$ 
    using set-concat that by fastforce
  have  $last\ a < hd\ (concat\ as)$ 
    using Cons.prems that by (auto simp: less-list-def)
  also have  $\dots \leq hd\ y$  if  $y \neq []$ 
    using that a
  by (meson  $\S$  order.strict-trans less-list-def not-le sorted-hd-le strict-sorted-imp-sorted)
  finally show ?thesis

```

```

    by (simp add: less-list-def)
  qed
  then show ?case
    using Cons by (auto simp: strict-sorted-append-iff)
  qed auto

proposition merge-exists:
  assumes strict-sorted (concat as) strict-sorted (concat bs)
    as ∈ lists (- {}) bs ∈ lists (- {})
    hd as < hd bs as ≠ [] bs ≠ []
  and disj:  $\bigwedge a b. [a \in \text{list.set } as; b \in \text{list.set } bs] \implies a < b \vee b < a$ 
  shows  $\exists us \text{ vs. merge } as \ bs \ us \ vs$ 
  using assms
  proof (induction length as + length bs arbitrary: as bs rule: less-induct)
  case (less as bs)
  obtain as1 as2 bs1 bs2
    where A: as1 ≠ [] bs1 ≠ [] concat as1 < concat bs1 concat bs1 < concat as2
      and B: as = as1@as2 bs = bs1@bs2 and C: bs2 = []  $\vee$  (as2 ≠ []  $\wedge$  hd as2
    < hd bs2)
  proof
  define as1 where as1  $\equiv$  takeWhile ( $\lambda x. x < \text{hd } bs$ ) as
  define as2 where as2  $\equiv$  dropWhile ( $\lambda x. x < \text{hd } bs$ ) as
  define bs1 where bs1  $\equiv$  if as2=[] then bs else takeWhile ( $\lambda x. x < \text{hd } as2$ ) bs
  define bs2 where bs2  $\equiv$  if as2=[] then [] else dropWhile ( $\lambda x. x < \text{hd } as2$ ) bs

  have as1: as1 = takeWhile ( $\lambda x. \text{last } x < \text{hd } (\text{hd } bs)$ ) as
    using less.prem1 by (auto simp: as1-def less-list-def cong: takeWhile-cong)
  have as2: as2 = dropWhile ( $\lambda x. \text{last } x < \text{hd } (\text{hd } bs)$ ) as
    using less.prem2 by (auto simp: as2-def less-list-def cong: dropWhile-cong)

  have hd-as2: as2 ≠ []  $\implies \neg \text{hd } as2 < \text{hd } bs$ 
    using as2-def hd-dropWhile by metis
  have hd-bs2: bs2 ≠ []  $\implies \neg \text{hd } bs2 < \text{hd } as2$ 
    using bs2-def hd-dropWhile by metis
  show as1 ≠ []
    by (simp add: as1-def less.prem1 takeWhile-eq-Nil-iff)
  show bs1 ≠ []
    by (metis as2 bs1-def hd-as2 hd-in-set less.prem1(7) less.prem1(8) set-dropWhileD
    takeWhile-eq-Nil-iff)
  show bs2 = []  $\vee$  (as2 ≠ []  $\wedge$  hd as2 < hd bs2)
    by (metis as2-def bs2-def hd-bs2 less.prem1(8) list.set-sel(1) set-dropWhileD)
  have AB: list.set A  $\ll$  list.set B
    if A ∈ list.set as1 B ∈ list.set bs for A B
  proof -
  have A ∈ list.set as
    using that by (metis as1 set-takeWhileD)
  then have sorted A
    by (metis concat.simps(2) concat-append less.prem1(1) sorted-append
    split-list-last strict-sorted-imp-sorted)

```

```

moreover have sorted (hd bs)
by (metis concat.simps(2) hd-Cons-tl less.prem(2) less.prem(7) strict-sorted-append-iff
strict-sorted-imp-sorted)
ultimately show ?thesis
using that less.prem
apply (clarsimp simp add: as1-def set-takeWhile less-list-iff-less-sets less-sets-def)
by (metis (full-types) UN-I hd-concat less-le-trans list.set-sel(1) set-concat
sorted-hd-le strict-sorted-imp-sorted)
qed
show as = as1@as2
by (simp add: as1-def as2-def)
show bs = bs1@bs2
by (simp add: bs1-def bs2-def)
have list.set (concat as1)  $\ll$  list.set (concat bs1)
using AB set-takeWhileD by (fastforce simp: as1-def bs1-def less-sets-UN1
less-sets-UN2)
then show concat as1 < concat bs1
by (rule less-sets-imp-list-less)
have list.set (concat bs1)  $\ll$  list.set (concat as2) if as2  $\neq$  []
proof (clarsimp simp add: bs1-def less-sets-UN1 less-sets-UN2 set-takeWhile
less.prem)
fix A B
assume A  $\in$  list.set as2 B  $\in$  list.set bs B < hd as2
with that show list.set B  $\ll$  list.set A
using hd-as2 less.prem(1,2)
apply (clarsimp simp add: less-sets-def less-list-def)
apply (auto simp: as2-def)
apply (simp flip: as2-def)
by (smt (verit, ccfv-SIG) UN-I  $\langle$ as = as1 @ as2 $\rangle$  concat.simps(2) con-
cat-append hd-concat in-set-conv-decomp-first le-less-trans less-le-trans set-concat
sorted-append sorted-hd-le sorted-le-last strict-sorted-imp-sorted that)
qed
then show concat bs1 < concat as2
by (simp add: bs1-def less-sets-imp-list-less)
qed
obtain cs ds where merge as2 bs2 cs ds
proof (cases as2 = []  $\vee$  bs2 = [])
case True
then show thesis
using that C NullNull Null by metis
next
have †: length as2 + length bs2 < length as + length bs
by (simp add: A B)
case False
moreover have strict-sorted (concat as2) strict-sorted (concat bs2)
as2  $\in$  lists (- {[]}) bs2  $\in$  lists (- {[]})
 $\wedge$  a b. [a  $\in$  list.set as2; b  $\in$  list.set bs2]  $\implies$  a < b  $\vee$  b < a
using B less.prem strict-sorted-append-iff by auto
ultimately show ?thesis

```

```

    using C less.hyps [OF †] False that by force
  qed
  then obtain cs where merge (as1 @ as2) (bs1 @ bs2) (concat as1 # cs) (concat
bs1 # ds)
    using A merge.App by blast
  then show ?case
    using B by blast
  qed

```

### 3.11.4 Actual proof of Larson's Lemma 3.8

**proposition** *lemma-3-8*:

**assumes** *infinite* N

**obtains** X **where**  $X \subseteq WW$  *ordertype* X (*lenlex less-than*) =  $\omega \uparrow \omega$

$\bigwedge u. u \in [X]^2 \implies$

$\exists l. \text{Form } l \ u \wedge (l > 0 \longrightarrow [\text{enum } N \ l] < \text{inter-scheme } l \ u \wedge \text{List.set}$

$(\text{inter-scheme } l \ u) \subseteq N)$

**proof** –

**let** ?LL = *lenlex less-than*

**define** bf **where**  $bf \equiv \lambda M \ q. \text{wfrec pair-less } (\lambda f \ (j, i).$

$\text{let } R = (\text{case prev } j \ i \ \text{of None} \Rightarrow M \mid \text{Some } u \Rightarrow \text{snd } (f$

$u))$

$\text{in grab } R \ (q \ j \ i))$

**have** bf-rec:  $bf \ M \ q \ (j, i) =$

$(\text{let } R = (\text{case prev } j \ i \ \text{of None} \Rightarrow M \mid \text{Some } u \Rightarrow \text{snd } (bf \ M \ q \ u))$

$\text{in grab } R \ (q \ j \ i)) \ \text{for } M \ q \ j \ i$

**by** (*subst* (1) *bf-def*) (*simp add: Let-def wfrec bf-def cut-apply prev-pair-less cong: conj-cong split: option.split*)

**have** *infinite* ( $\text{snd } (bf \ M \ q \ u) = \text{infinite } M \wedge \text{fst } (bf \ M \ q \ u) \subseteq M \wedge \text{snd } (bf \ M$

$q \ u) \subseteq M$  **for**  $M \ q \ u$

**using** *wf-pair-less*

**proof** (*induction* u *rule: wf-induct-rule*)

**case** (*less* u)

**then show** ?case

**proof** (*cases* u)

**case** (*Pair* j i)

**with** *less.IH prev-pair-less* **show** ?thesis

**apply** (*simp add: bf-rec [of M q j i] split: option.split*)

**using** *fst-grab-subset snd-grab-subset* **by** *blast*

**qed**

**qed**

**then have** *infinite-bf* [*simp*]:  $\text{infinite } (\text{snd } (bf \ M \ q \ u)) = \text{infinite } M$

**and** *bf-subset*:  $\text{fst } (bf \ M \ q \ u) \subseteq M \wedge \text{snd } (bf \ M \ q \ u) \subseteq M$  **for**  $M \ q \ u$

**by** *auto*

**have** *bf-less-sets*:  $\text{fst } (bf \ M \ q \ ij) \ll \text{snd } (bf \ M \ q \ ij)$  **if** *infinite* M **for**  $M \ q \ ij$

**using** *wf-pair-less*

```

proof (induction ij rule: wf-induct-rule)
  case (less u)
  then show ?case
    by (metis bf-rec finite-grab-iff infinite-bf less-sets-grab prod.exhaust-sel that)
qed

have card-fst-bf: finite (fst (bf M q (j,i))) ∧ card (fst (bf M q (j,i))) = q j i if
infinite M for M q j i
  by (simp add: that bf-rec [of M q j i] split: option.split)

have bf-cong: bf M q u = bf M q' u
  if snd u ≤ fst u and eq: ∧y x. [x ≤ y; y ≤ fst u] ⇒ q' y x = q y x for M q q' u
  using wf-pair-less that
proof (induction u rule: wf-induct-rule)
  case (less u)
  show ?case
  proof (cases u)
  case (Pair j i)
  with less.premis show ?thesis
  proof (clarsimp simp add: bf-rec [of M - j i] split: option.split)
  fix j' i'
  assume *: prev j i = Some (j',i')
  then have **: ((j', i'), u) ∈ pair-less
    by (simp add: Pair prev-pair-less)
  moreover have i' < j'
    using Pair less.premis by (simp add: prev-Some-less [OF *])
  moreover have ∧x y. [x ≤ y; y ≤ j'] ⇒ q' y x = q y x
    using ** less.premis by (auto simp: pair-less-def Pair)
  ultimately show grab (snd (bf M q (j',i'))) (q j i) = grab (snd (bf M q'
(j',i'))) (q j i)
    using less.IH by auto
  qed
qed
qed

define ediff where ediff ≡ λD:: nat ⇒ nat set. λj i. enum (D j) (Suc i) – enum
(D j) i
define F where F ≡ λl (dl,a0::nat set,b0::nat × nat ⇒ nat set,M).
  let (d,Md) = grab (nxt M (enum N (Suc (2 * Suc l)))) (Suc l) in
  let (a,Ma) = grab Md (Min d) in
  let Gb = bf Ma (ediff (dl(l := d))) in
  let dl' = dl(l := d) in
  (dl', a, fst ∘ Gb, snd (Gb(l, l-1)))
define DF where DF ≡ rec-nat (λi∈{..<0}. {}, {}, λp. {}, N) F
have DF-simps: DF 0 = (λi∈{..<0}. {}, {}, λp. {}, N)
  DF (Suc l) = F l (DF l) for l
  by (auto simp: DF-def)
note cut-apply [simp]

```

**have** *inf [rule-format]:  $\forall dl\ al\ bl\ L. DF\ l = (dl,al,bl,L) \longrightarrow infinite\ L$  for  $l$*   
**by** (*induction l*) (*auto simp: DF-simps F-def Let-def grab-eqD infinite-nxtN*  
*assms split: prod.split*)

**define**  $\Psi$  **where**  
 $\Psi \equiv \lambda(dl, a, b, M). \lambda l::nat.$   
 $dl\ l \ll a \wedge card\ a > 0 \wedge$   
 $(\forall j \leq l. card\ (dl\ j) = Suc\ j) \wedge a \ll \bigcup (range\ b) \wedge range\ b \subseteq Collect\ finite$   
 $\wedge$   
 $a \subseteq N \wedge \bigcup (range\ b) \subseteq N \wedge infinite\ M \wedge b(l,l-1) \ll M \wedge M \subseteq N$

**have**  $\Psi$ -DF:  $\Psi\ (DF\ (Suc\ l))\ l$  **for**  $l$   
**proof** (*induction l*)  
**case** 0  
**show** ?case  
**using** *assms*  
**apply** (*clarsimp simp add: bf-rec F-def DF-simps  $\Psi$ -def split: prod.split*)  
**apply** (*drule grab-eqD, blast dest: grab-eqD infinite-nxtN*)+  
**apply** (*auto simp: less-sets-UN2 less-sets-grab card-fst-bf elim!: less-sets-weaken2*)  
**apply** (*metis card-1-singleton-iff Min-singleton greaterThan-iff insertI1 le0*  
*nxt-subset-greaterThan subsetD*)  
**using** *nxt-subset snd-grab-subset bf-subset* **by** *blast*+  
**next**  
**case** (*Suc l*)  
**then show** ?case  
**using** *assms*  
**unfolding** *Let-def DF-simps(2)[of Suc l] F-def  $\Psi$ -def*  
**apply** (*clarsimp simp add: bf-rec DF-simps split: prod.split*)  
**apply** (*drule grab-eqD, metis grab-eqD infinite-nxtN*)+  
**apply** (*safe, simp-all add: less-sets-UN2 less-sets-grab card-fst-bf card-Suc-eq-finite*)  
**apply** (*meson less-sets-weaken2*)  
**apply** (*metis Min-in grOI greaterThan-iff insert-not-empty le-inf-iff*  
*less-asm nxt-def subsetD*)  
**apply** (*meson bf-subset less-sets-weaken2*)  
**apply** (*meson nxt-subset subset-eq*)  
**apply** (*meson bf-subset nxt-subset subset-eq*)  
**using** *bf-rec infinite-bf* **apply** *force*  
**using** *bf-less-sets bf-rec* **apply** *force*  
**by** (*metis bf-rec bf-subset nxt-subset subsetD*)  
**qed**

**define**  $d$  **where**  $d \equiv \lambda k. let\ (dk,ak,bk,M) = DF(Suc\ k)\ in\ dk\ k$   
**define**  $a$  **where**  $a \equiv \lambda k. let\ (dk,ak,bk,M) = DF(Suc\ k)\ in\ ak$   
**define**  $b$  **where**  $b \equiv \lambda k. let\ (dk,ak,bk,M) = DF(Suc\ k)\ in\ bk$   
**define**  $M$  **where**  $M \equiv \lambda k. let\ (dk,ak,bk,M) = DF\ k\ in\ M$

**have** *infinite-M [simp]: infinite (M k) for k*  
**by** (*auto simp: M-def inf split: prod.split*)

**have** *M-Suc-subset: M (Suc k)  $\subseteq$  M k for k*

**apply** (*clarsimp simp add: Let-def M-def F-def DF-simps split: prod.split*)  
**by** (*metis bf-subset in-mono nxt-subset snd-conv snd-grab-subset*)

**have** *Inf-M-Suc-ge*:  $\text{Inf } (M \ k) \leq \text{Inf } (M \ (\text{Suc } k))$  **for**  $k$   
**by** (*simp add: M-Suc-subset cInf-superset-mono infinite-imp-nonempty*)

**have** *Inf-M-telescoping*:  $\{\text{Inf } (M \ k)..\} \subseteq \{\text{Inf } (M \ k')..\}$  **if**  $k': k' \leq k$  **for**  $k \ k'$   
**using** *that Inf-nat-def1 infinite-M unfolding Inf-nat-def atLeast-subset-iff*  
**by** (*metis M-Suc-subset finite.emptyI le-less-linear lift-Suc-antimono-le not-less-Least subsetD*)

**have** *d-eq*:  $d \ k = \text{fst } (\text{grab } (\text{nxt } (M \ k) \ (\text{enum } N \ (\text{Suc } (2 * \text{Suc } k)))) \ (\text{Suc } k))$  **for**  $k$   
**by** (*simp add: d-def M-def Let-def DF-simps F-def split: prod.split*)  
**then have** *finite-d* [*simp*]:  $\text{finite } (d \ k)$  **for**  $k$   
**by** *simp*  
**then have** *d-ne* [*simp*]:  $d \ k \neq \{\}$  **for**  $k$   
**by** (*metis card.empty card-grab d-eq infinite-M infinite-nxtN nat.distinct(1)*)  
**have** *a-eq*:  $\exists M. a \ k = \text{fst } (\text{grab } M \ (\text{Min } (d \ k))) \wedge \text{infinite } M$  **for**  $k$   
**apply** (*simp add: a-def d-def M-def Let-def DF-simps F-def split: prod.split*)  
**by** (*metis fst-conv grab-eqD infinite-nxtN local.inf*)  
**then have** *card-a*:  $\text{card } (a \ k) = \text{Inf } (d \ k)$  **for**  $k$   
**by** (*metis cInf-eq-Min card-grab d-ne finite-d*)

**have** *d-eq-dl*:  $d \ k = dl \ k$  **if**  $(dl, a, b, P) = DF \ l \ k < l$  **for**  $k \ l \ dl \ a \ b \ P$   
**using** *that*  
**by** (*induction l arbitrary: dl a b P*) (*simp-all add: d-def DF-simps F-def Let-def split: prod.split-asm prod.split*)

**have** *card-d* [*simp*]:  $\text{card } (d \ k) = \text{Suc } k$  **for**  $k$   
**by** (*auto simp: d-eq infinite-nxtN*)

**have** *d-ne* [*simp*]:  $d \ j \neq \{\}$  **and** *a-ne* [*simp*]:  $a \ j \neq \{\}$   
**and** *finite-d* [*simp*]:  $\text{finite } (d \ j)$  **and** *finite-a* [*simp*]:  $\text{finite } (a \ j)$  **for**  $j$   
**using**  $\Psi\text{-DF}$  [*of j*] **by** (*auto simp:  $\Psi$ -def a-def d-def card-gt-0-iff split: prod.split-asm*)

**have** *da*:  $d \ k \ll a \ k$  **for**  $k$   
**using**  $\Psi\text{-DF}$  [*of k*] **by** (*simp add:  $\Psi$ -def a-def d-def split: prod.split-asm*)

**have** *ab-same*:  $a \ k \ll \bigcup (\text{range } (b \ k))$  **for**  $k$   
**using**  $\Psi\text{-DF}$  [*of k*] **by** (*simp add:  $\Psi$ -def a-def b-def M-def split: prod.split-asm*)

**have** *snd-bf-subset*:  $\text{snd } (bf \ M \ r \ (j, i)) \subseteq \text{snd } (bf \ M \ r \ (j', i'))$   
**if**  $ji: ((j', i'), (j, i)) \in \text{pair-less } (j', i') \in IJ \ k$   
**for**  $M \ r \ k \ j \ i \ j' \ i'$   
**using** *wf-pair-less ji*  
**proof** (*induction rule: wf-induct-rule [where a = (j, i)]*)  
**case** (*less u*)  
**show** *?case*

```

proof (cases u)
  case (Pair j i)
    then consider prev j i = Some (j', i') | x where ((j', i'), x) ∈ pair-less prev
j i = Some x
    using less.premis pair-less-prev by blast
    then show ?thesis
    proof cases
      case 2 with less.IH show ?thesis
      unfolding bf-rec Pair
      by (metis in-mono option.simps(5) prev-pair-less snd-grab-subset subsetI
that(2))
    qed (simp add: Pair bf-rec snd-grab-subset)
  qed
qed

have less-bf: fst (bf M r (j',i')) ≤ fst (bf M r (j,i))
  if ji: ((j',i'), (j,i)) ∈ pair-less (j',i') ∈ IJ k and infinite M
  for M r k j i j' i'
proof -
  consider prev j i = Some (j', i') | j'' i'' where ((j', i'), (j'',i'')) ∈ pair-less
prev j i = Some (j'',i'')
  by (metis pair-less-prev ji prod.exhaust-sel)
  then show ?thesis
  proof cases
    case 1
      then show ?thesis
      using bf-less-sets bf-rec less-sets-fst-grab ⟨infinite M⟩ by force
    next
      case 2
        then have fst (bf M r (j',i')) ≤ snd (bf M r (j'',i''))
        by (meson bf-less-sets snd-bf-subset less-sets-weaken2 that)
        with 2 show ?thesis
        using bf-rec bf-subset less-sets-fst-grab ⟨infinite M⟩ by auto
  qed
qed

have aM: a k ≤ M (Suc k) for k
  apply (clarsimp simp add: a-def M-def DF-simps F-def Let-def split: prod.split)
  by (meson bf-subset grab-eqD infinite-nxtN less-sets-weaken2 local.inf)
then have a k ≤ a (Suc k) for k
  by (metis IntE card-d card.empty d-eq da fst-grab-subset less-sets-trans less-sets-weaken2
nat.distinct(1) nxt-def subsetI)
then have aa: a j ≤ a k if j < k for k j
  by (meson UNIV-I a-ne less-sets-imp-strict-mono-sets strict-mono-sets-def that)
then have ab: a k' ≤ b k (j,i) if k' ≤ k for k k' j i
  by (metis a-ne ab-same le-less less-sets-UN2 less-sets-trans rangeI that)
have db: d j ≤ b k (j,i) if j ≤ k for k j i
  by (meson a-ne ab da less-sets-trans that)

```



```

have bMkk: b k (k,k-1) << M (Suc k) for k
  using Ψ-DF [of k]
  by (simp add: Ψ-def b-def d-def M-def split: prod.split-asm)

have b: ∃ P ⊆ M k. infinite P ∧ (∀ j i. i ≤ j → j ≤ k → b k (j,i) = fst (bf P
(ediff d) (j,i))) for k
proof (clarsimp simp: b-def DF-simps F-def Let-def split: prod.split)
  fix a a' d' dl bb P M' M''
  assume gr: grab M'' (Min d') = (a', M') grab (nxt P (enum N (Suc (Suc (Suc
(2 * k)))))) (Suc k) = (d', M'')
  and DF: DF k = (dl, a, bb, P)
  have deg: d j = (if j = k then d' else dl j) if j ≤ k for j
  proof (cases j < k)
    case True
      then show ?thesis by (metis DF d-eq-dl less-not-refl)
    next
      case False
        then show ?thesis
          using that DF gr by (auto simp: d-def DF-simps F-def Let-def split: prod.split)
  qed
  have M' ⊆ P
    by (metis gr in-mono nxt-subset snd-conv snd-grab-subset subsetI)
  also have P ⊆ M k
    using DF by (simp add: M-def)
  finally have M' ⊆ M k .
  moreover have infinite M'
    using DF by (metis (mono-tags) finite-grab-iff gr infinite-nxtN local.inf
snd-conv)
  moreover
  have ediff (dl(k := d')) j i = ediff d j i if j ≤ k for j i
    by (simp add: deg that ediff-def)
  then have bf M' (ediff (dl(k := d'))) (j,i)
    = bf M' (ediff d) (j,i) if i ≤ j j ≤ k for j i
    using bf-cong that by fastforce
  ultimately show ∃ P ⊆ M k. infinite P ∧
    (∀ j i. i ≤ j → j ≤ k
      → fst (bf M' (ediff (dl(k := d'))) (j,i))
      = fst (bf P (ediff d) (j,i)))

    by auto
qed

have card-b: card (b k (j,i)) = enum (d j) (Suc i) - enum (d j) i if j ≤ k for k j
i
  — there's a short proof of this from the previous result but it would need i ≤ j
proof (clarsimp simp: b-def DF-simps F-def Let-def split: prod.split)
  fix dl
  and a a' d':: nat set
  and bb M M' M''
  assume gr: grab M'' (Min d') = (a', M') grab (nxt M (enum N (Suc (Suc (Suc

```

```

(2 * k)))))) (Suc k) = (d', M'')
  and DF: DF k = (dl, a, bb, M)
  have d j = (if j = k then d' else dl j)
  proof (cases j < k)
    case True
      then show ?thesis by (metis DF d-eq-dl less-not-refl)
    next
      case False
        then show ?thesis
          using that DF gr by (auto simp: d-def DF-simps F-def Let-def split: prod.split)
  qed
  then show card (fst (bf M' (ediff (dl(k := d')) (j, i)))
    = enum (d j) (Suc i) - enum (d j) i
    using DF gr card-fst-bf grab-eqD infinite-nextN local.inf ediff-def by auto
qed

have card-b-pos: card (b k (j, i)) > 0 if i < j j ≤ k for k j i
  by (simp add: card-b that finite-enumerate-step)
have b-ne [simp]: b k (j, i) ≠ {} if i < j j ≤ k for k j i
  using card-b-pos [OF that] less-imp-neq by fastforce+

have card-b-finite [simp]: finite (b k u) for k u
  using Ψ-DF [of k] by (fastforce simp: Ψ-def b-def)

have bM: b k (j, i) ≪ M (Suc k) if i < j j ≤ k for i j k
proof -
  obtain M' where M' ⊆ M k infinite M'
  and bk: ∧ j i. i ≤ j ⇒ j ≤ k ⇒ b k (j, i) = fst (bf M' (ediff d) (j, i))
  using b by (metis (no-types, lifting))
  show ?thesis
  proof (cases j = k ∧ i = k - 1)
    case False
      show ?thesis
      proof (rule less-sets-trans [OF - bMkk])
        show b k (j, i) ≪ b k (k, k - 1)
          using that ‹infinite M'› False
          by (force simp: bk pair-less-def IJ-def intro: less-bf)
        show b k (k, k - 1) ≠ {}
          using b-ne that by auto
      qed
    qed (use bMkk in auto)
  qed

have b-InfM: ⋃ (range (b k)) ⊆ {⋂ (M k)..} for k
  proof (clarisimp simp add: Ψ-def b-def M-def DF-simps F-def Let-def split:
  prod.split)
    fix r dl :: nat ⇒ nat set
    and a b and d' a' M'' M' P and x j' i' :: nat
    assume gr: grab M'' (Min d') = (a', M')

```

grab (nxt P (enum N (Suc (Suc (Suc (2 \* k)))))) (Suc k) = (d', M'')  
 and DF: DF k = (dl, a, b, P)  
 and x: x ∈ fst (bf M' (ediff (dl(k := d'))) (j', i'))  
 have infinite P  
 using DF local.inf by blast  
 then have M' ⊆ P  
 by (meson gr grab-eqD infinite-nxtN nxt-subset order.trans)  
 with bf-subset show □ P ≤ x  
 using Inf-nat-def x le-less-linear not-less-Least by fastforce  
 qed

have b-Inf-M-Suc: b k (j, i) ≪ {Inf(M (Suc k))} if i < j j ≤ k for k j i  
 using bMkk [of k] that  
 by (metis Inf-nat-def1 bM finite.emptyI infinite-M less-setsD less-sets-singleton2)

have bb-same: b k (j', i') ≪ b k (j, i)  
 if ((j', i'), (j, i)) ∈ pair-less (j', i') ∈ IJ k for k j i j' i'  
 using that  
 unfolding b-def DF-simps F-def Let-def  
 by (auto simp: less-bf grab-eqD infinite-nxtN local.inf split: prod.split)

have bb: b k' (j', i') ≪ b k (j, i)  
 if j: i' < j' j' ≤ k' and k: k' < k for i i' j j' k' k  
 proof (rule atLeast-less-sets)  
 show b k' (j', i') ≪ {Inf(M (Suc k'))}  
 using Suc-lessD b-Inf-M-Suc nat-less-le j by blast  
 show b k (j, i) ⊆ {Inf(M (Suc k'))..<}  
 by (meson Inf-M-telescoping Suc-leI UnionI b-InfM rangeI subset-eq k)  
 qed

have M-subset-N: M k ⊆ N for k  
 proof (cases k)  
 case (Suc k')  
 with Ψ-DF [of k'] show ?thesis  
 by (auto simp: M-def Let-def Ψ-def split: prod.split)  
 qed (auto simp: M-def DF-simps)  
 have a-subset-N: a k ⊆ N for k  
 using Ψ-DF [of k] by (simp add: a-def Ψ-def split: prod.split prod.split-asm)  
 have d-subset-N: d k ⊆ N for k  
 using M-subset-N [of k] d-eq fst-grab-subset nxt-subset by blast  
 have b-subset-N: b k (j, i) ⊆ N for k j i  
 using Ψ-DF [of k] by (force simp: b-def Ψ-def)

define K:: [nat, nat] ⇒ nat set set  
 where K ≡ λj0 j. nsets {j0 < ..} j  
 have K-finite: finite K and K-card: card K = j if K ∈ K j0 j for K j0 j  
 using that by (auto simp add: K-def nsets-def)  
 have K-enum: j0 < enum K i if K ∈ K j0 j i < card K for K j0 j i  
 using that by (auto simp: K-def nsets-def finite-enumerate-in-set subset-eq)

**have**  $\mathcal{K}\text{-}0$  [simp]:  $\mathcal{K} \ k \ 0 = \{\{\}\}$  **for**  $k$   
**by** (auto simp:  $\mathcal{K}\text{-def}$ )

**have**  $\mathcal{K}\text{-Suc}$ :  $\mathcal{K} \ j0 \ (Suc \ j) = USigma \ (\mathcal{K} \ j0 \ j) \ (\lambda K. \ \{Max \ (insert \ j0 \ K) <..\})$  (is  
 $?lhs = ?rhs$ )  
**for**  $j \ j0$   
**proof**  
**show**  $\mathcal{K} \ j0 \ (Suc \ j) \subseteq USigma \ (\mathcal{K} \ j0 \ j) \ (\lambda K. \ \{Max \ (insert \ j0 \ K) <..\})$   
**unfolding**  $\mathcal{K}\text{-def}$   $nsets\text{-def}$   $USigma\text{-def}$   
**proof** clarsimp  
**fix**  $K$   
**assume**  $K: K \subseteq \{j0 <..\}$   $finite \ K$   $card \ K = Suc \ j$   
**then have**  $Max \ K \in K$   
**by** (metis  $Max\text{-in}$   $card\text{-}0\text{-eq}$   $nat.\text{distinct}(1)$ )  
**then obtain**  $i$  **where**  $Max \ (insert \ j0 \ (K - \{Max \ K\})) < i \ K = insert \ i \ (K - \{Max \ K\})$   
**using**  $K$   
**by** (simp add:  $subset\text{-iff}$ ) (metis  $DiffE$   $Max.\text{cobounded}I$   $insertCI$   $insert\text{-Diff}$   $le\text{-neq}\text{-implies}\text{-less}$ )  
**then show**  $\exists L \subseteq \{j0 <..\}. \ finite \ L \wedge \ card \ L = j \wedge (\exists i \in \{Max \ (insert \ j0 \ L) <..\}. \ K = insert \ i \ L)$   
**using**  $K$   
**by** (metis  $\langle Max \ K \in K \rangle \ card\text{-Diff}\text{-singleton}\text{-if}$   $diff\text{-Suc}\text{-}1$   $finite\text{-Diff}$   $greaterThan\text{-iff}$   $insert\text{-subset}$ )  
**qed**  
**show**  $?rhs \subseteq \mathcal{K} \ j0 \ (Suc \ j)$   
**by** (force simp:  $\mathcal{K}\text{-def}$   $nsets\text{-def}$   $USigma\text{-def}$ )  
**qed**

**define**  $BB$  **where**  $BB \equiv \lambda j0 \ j \ K. \ list\text{-of} \ (a \ j0 \cup (\bigcup i < j. \ b \ (enum \ K \ i) \ (j0, i)))$   
**define**  $XX$  **where**  $XX \equiv \lambda j. \ BB \ j \ j \ ' \ \mathcal{K} \ j \ j$

**have**  $less\text{-list}\text{-of}$ :  $BB \ j \ i \ K < list\text{-of} \ (b \ l \ (j, i))$   
**if**  $K: K \in \mathcal{K} \ j \ i \ \forall j \in K. \ j < l$  **and**  $i \leq j \ j \leq l$  **for**  $j \ i \ K \ l$   
**unfolding**  $BB\text{-def}$   
**proof** (rule  $less\text{-sets}\text{-imp}\text{-sorted}\text{-list}\text{-of}\text{-set}$ )  
**have**  $\bigwedge i. \ i < card \ K \implies b \ (enum \ K \ i) \ (j, i) \ll b \ l \ (j, card \ K)$   
**using that** **by** (metis  $\mathcal{K}\text{-card}$   $\mathcal{K}\text{-enum}$   $\mathcal{K}\text{-finite}$   $bb$   $finite\text{-enumerate}\text{-in}\text{-set}$   $nat\text{-less}\text{-le}$   $less\text{-le}\text{-trans}$ )  
**then show**  $a \ j \cup (\bigcup i < i. \ b \ (enum \ K \ i) \ (j, i)) \ll b \ l \ (j, i)$   
**using that** **unfolding**  $\mathcal{K}\text{-def}$   $nsets\text{-def}$   
**by** (auto simp:  $less\text{-sets}\text{-Un}1$   $less\text{-sets}\text{-UN}1$   $ab$   $finite\text{-enumerate}\text{-in}\text{-set}$   $subset\text{-eq}$ )  
**qed** auto  
**have**  $BB\text{-Suc}$ :  $BB \ j0 \ (Suc \ j) \ K = usplit \ (\lambda L \ k. \ BB \ j0 \ j \ L \ @ \ list\text{-of} \ (b \ k \ (j0, j)))$   
 $K$   
**if**  $j: j \leq j0$  **and**  $K: K \in \mathcal{K} \ j0 \ (Suc \ j)$  **for**  $j0 \ j \ K$   
— towards the ordertype proof  
**proof** —  
**have**  $Kj$ :  $K \subseteq \{j0 <..\}$  **and** [simp]:  $finite \ K$  **and**  $cardK$ :  $card \ K = Suc \ j$

```

    using K by (auto simp: K-def nsets-def)
  have KMK:  $K - \{Max\ K\} \in \mathcal{K}\ j0\ j$ 
    using that by (simp add: K-Suc USigma-iff K-finite less-sets-def usplit-def)
  have  $j0 < Max\ K$ 
    by (metis Kj Max-in cardK card-gt-0-iff greaterThan-iff subsetD zero-less-Suc)
  have MaxK:  $Max\ K = enum\ K\ j$ 
  proof (rule Max-eqI)
    fix k
    assume  $k \in K$ 
    with K cardK show  $k \leq enum\ K\ j$ 
      by (metis ⟨finite K⟩ finite-enumerate-Ex finite-enumerate-mono-iff leI lessI
not-less-eq)
    qed (auto simp: cardK finite-enumerate-in-set)
  have ene:  $i < j \implies enum\ (K - \{enum\ K\ j\})\ i = enum\ K\ i$  for i
    using finite-enumerate-Diff-singleton [OF ⟨finite K⟩] by (simp add: cardK)
  have BB j0 (Suc j)  $K = list-of\ ((a\ j0 \cup (\bigcup_{x < j}. b\ (enum\ K\ x)\ (j0, x))) \cup b$ 
  (enum K j) (j0, j))
    by (simp add: BB-def lessThan-Suc Un-ac)
  also have ... = list-of ((a j0  $\cup (\bigcup_{i < j}. b\ (enum\ K\ i)\ (j0, i))$ )) @ list-of (b
  (enum K j) (j0, j))
  proof (rule sorted-list-of-set-Un)
    have  $b\ (enum\ K\ i)\ (j0, i) \ll b\ (enum\ K\ j)\ (j0, j)$  if  $i < j$  for i
      using K K-enum bb cardK j le-eq-less-or-eq that by auto
    moreover have  $a\ j0 \ll b\ (enum\ K\ j)\ (j0, j)$ 
      using MaxK ⟨j0 < Max K⟩ ab by auto
    ultimately show  $a\ j0 \cup (\bigcup_{x < j}. b\ (enum\ K\ x)\ (j0, x)) \ll b\ (enum\ K\ j)\ (j0,$ 
j)
      by (simp add: less-sets-Un1 less-sets-UN1)
    qed (auto simp: finite-UnI)
  also have ... = BB j0 j  $(K - \{Max\ K\})$  @ list-of (b (Max K) (j0, j))
    by (simp add: BB-def MaxK ene)
  also have ... = usplit ( $\lambda L\ k. BB\ j0\ j\ L$  @ list-of (b k (j0, j))) K
    by (simp add: usplit-def)
  finally show ?thesis .
qed

have enum-d-0:  $enum\ (d\ j)\ 0 = Inf\ (d\ j)$  for j
  using enum-0-eq-Inf-finite by auto

have Inf-b-less:  $\prod (b\ k'\ (j', i')) < \prod (b\ k\ (j, i))$ 
  if j:  $i' < j'$   $i < j$   $j' \leq k'$   $j \leq k$  and k:  $k' < k$  for  $i\ i'\ j\ j'\ k'\ k$ 
  using bb [of i' j' k' k j i] that b-ne [of i' j' k'] b-ne [of i j k]
  by (simp add: less-sets-def Inf-nat-def1)

have b-ge-k:  $\prod (b\ k\ (k, k-1)) \geq k-1$  for k
  proof (induction k)
    case (Suc k)
    show ?case
    proof (cases k=0)

```

```

case False
then have  $\prod (b\ k\ (k, k - 1)) < \prod (b\ (Suc\ k)\ (Suc\ k, k))$ 
  using Inf-b-less by auto
with Suc show ?thesis
  by simp
qed auto
qed auto

have b-ge:  $\prod (b\ k\ (j, i)) \geq k - 1$  if  $k \geq j\ j > i$  for  $k\ j\ i$ 
  by (metis Inf-b-less Suc-leI b-ge-k diff-Suc-1 lessI not-less that diff-le-mono)
have hd-b:  $hd\ (list-of\ (b\ k\ (j, i))) = \prod (b\ k\ (j, i))$ 
  if  $i < j\ j \leq k$  for  $k\ j\ i$ 
  using that by (simp add: hd-list-of cInf-eq-Min)

have b-disjoint-less:  $b\ (enum\ K\ i)\ (j0, i) \cap b\ (enum\ K\ i')\ (j0, i') = \{\}$ 
  if  $K: K \subseteq \{j0 <..\}$  finite  $K$   $card\ K \geq j0$   $i < j\ i' < j\ i < i'\ j \leq j0$  for  $i\ i'\ j\ j0\ K$ 
proof (intro bb less-sets-imp-disjnt [unfolded disjnt-def])
  show  $i < j0$ 
    using that by linarith
  then show  $j0 \leq enum\ K\ i$ 
    by (meson  $K$  finite-enumerate-in-set greaterThan-iff less-imp-le-nat less-le-trans
subsetD)
    show  $enum\ K\ i < enum\ K\ i'$ 
      using  $K\ \langle j \leq j0 \rangle$  that by auto
qed

have b-disjoint:  $b\ (enum\ K\ i)\ (j0, i) \cap b\ (enum\ K\ i')\ (j0, i') = \{\}$ 
  if  $K: K \subseteq \{j0 <..\}$  finite  $K$   $card\ K \geq j0$   $i < j\ i' < j\ i \neq i'\ j \leq j0$  for  $i\ i'\ j\ j0\ K$ 
  using that b-disjoint-less inf-commute neq-iff by metis

have otw: ordertype  $((\lambda k. list-of\ (b\ k\ (j, i)))\ \text{'}\ \{Max\ (insert\ j\ K) <..\})\ ?LL = \omega$ 
  (is ?lhs = -)
  if  $K: K \in \mathcal{K}\ j\ i\ j > i$  for  $j\ i\ K$ 
proof -
  have Sucj:  $Suc\ (Max\ (insert\ j\ K)) \geq j$ 
    using  $\mathcal{K}$ -finite that(1) le-Suc-eq by auto
  let  $?N = \{Inf(b\ k\ (j, i)) \mid k. Max\ (insert\ j\ K) < k\}$ 
  have infN: infinite  $?N$ 
proof (clarsimp simp add: infinite-nat-iff-unbounded-le)
  fix  $m$ 
  show  $\exists n \geq m. \exists k. n = \prod (b\ k\ (j, i)) \wedge Max\ (insert\ j\ K) < k$ 
    using b-ge  $\langle j > i \rangle$  Sucj
  by (metis (no-types, lifting) diff-Suc-1 le-SucI le-trans less-Suc-eq-le nat-le-linear)
qed
have [simp]:  $Max\ (insert\ j\ K) < k \iff j < k \wedge (\forall a \in K. a < k)$  for  $k$ 
  using that by (auto simp:  $\mathcal{K}$ -finite)
have ?lhs = ordertype  $?N$  less-than
proof (intro ordertype-eqI strip)
  have  $list-of\ (b\ k\ (j, i)) = list-of\ (b\ k'\ (j, i))$ 

```

**if**  $j \leq k$  **for**  $k$  **for**  $k'$   $hd (list-of (b k (j,i))) = hd (list-of (b k' (j,i)))$   
**for**  $k k'$   
**by** (*metis Inf-b-less*  $\langle i < j \rangle$  *hd-b nat-less-le not-le that*)  
**moreover have**  $\exists k' j' i'. hd (list-of (b k (j,i))) = \sqcap (b k' (j', i')) \wedge i' < j'$   
 $\wedge j' \leq k'$   
**if**  $j \leq k$  **for**  $k$   
**using that**  $\langle i < j \rangle$  *hd-b less-imp-le-nat* **by** *blast*  
**moreover have**  $\exists k'. hd (list-of (b k (j,i))) = \sqcap (b k' (j,i)) \wedge j < k' \wedge$   
 $(\forall a \in K. a < k')$   
**if**  $j < k \forall a \in K. a < k$  **for**  $k$   
**using that**  $K$  *hd-b less-imp-le-nat* **by** *blast*  
**moreover have**  $\sqcap (b k (j,i)) \in hd \text{ ' } (\lambda k. list-of (b k (j,i))) \text{ ' } \{Max (insert j$   
 $K) < ..\}$   
**if**  $j < k \forall a \in K. a < k$  **for**  $k$   
**using that**  $K$  **by** (*auto simp: hd-b image-iff*)  
**ultimately**  
**show** *bij-betw*  $hd ((\lambda k. list-of (b k (j,i))) \text{ ' } \{Max (insert j K) < ..\}) \{\sqcap (b k$   
 $(j,i)) | k. Max (insert j K) < k\}$   
**by** (*auto simp: bij-betw-def inj-on-def*)  
**next**  
**fix**  $ms ns$   
**assume**  $ms \in (\lambda k. list-of (b k (j,i))) \text{ ' } \{Max (insert j K) < ..\}$   
**and**  $ns \in (\lambda k. list-of (b k (j,i))) \text{ ' } \{Max (insert j K) < ..\}$   
**with that obtain**  $k k'$  **where**  
 $ms: ms = list-of (b k (j,i))$  **and**  $ns: ns = list-of (b k' (j,i))$   
**and**  $j < k$   $j < k'$  **and**  $lt-k: \forall a \in K. a < k$  **and**  $lt-k': \forall a \in K. a < k'$   
**by** (*auto simp: K-finite*)  
**then have** *len-eq [simp]: length ns = length ms*  
**by** (*simp add: card-b*)  
**have**  $nz: length ns \neq 0$   
**using** *b-ne*  $\langle i < j \rangle \langle j < k' \rangle$   $ns$  **by** *auto*  
**show**  $(hd ms, hd ns) \in less-than \longleftrightarrow (ms, ns) \in ?LL$   
**proof**  
**assume**  $(hd ms, hd ns) \in less-than$   
**then show**  $(ms, ns) \in ?LL$   
**using that**  $nz$   
**by** (*fastforce simp: lenlex-def K-finite card-b intro: hd-lex*)  
**next**  
**assume**  $\S: (ms, ns) \in ?LL$   
**then have**  $(list-of (b k' (j,i)), list-of (b k (j,i))) \notin ?LL$   
**using** *less-asym ms ns omega-sum-1-less* **by** *blast*  
**then show**  $(hd ms, hd ns) \in less-than$   
**using**  $\langle j < k \rangle \langle j < k' \rangle$  *Inf-b-less [of i j i j] ms ns*  
**by** (*metis Cons-lenlex-iff*  $\S$  *len-eq b-ne card-b-finite diff-Suc-1 hd-Cons-tl hd-b*  
*length-Cons less-or-eq-imp-le less-than-iff linorder-neqE-nat sorted-list-of-set-eq-Nil-iff*  
*that(2)*)  
**qed**  
**qed** *auto*  
**also have**  $\dots = \omega$

```

    using infN ordertype-nat- $\omega$  by blast
  finally show ?thesis .
qed

have otwj: ordertype (BB j0 j '  $\mathcal{K}$  j0 j) ?LL =  $\omega \uparrow j$  if  $j \leq j0$  for  $j j0$ 
  using that
proof (induction j) — a difficult proof, but no hints in Larson's text
  case 0
  then show ?case
    by (auto simp: XX-def)
next
  case (Suc j)
  then have ih: ordertype (BB j0 j '  $\mathcal{K}$  j0 j) ?LL =  $\omega \uparrow j$ 
    by simp
  have  $j \leq j0$ 
    by (simp add: Suc.prem1 Suc-leD)
  have inj-BB: inj-on (BB j0 j) ( $\{\{j0 < ..\}\}^j$ )
proof (clarsimp simp: inj-on-def BB-def nsets-def sorted-list-of-set-Un less-sets-UN2)
  fix X Y
  assume X:  $X \subseteq \{j0 < ..\}$  and Y:  $Y \subseteq \{j0 < ..\}$ 
    and finite X finite Y
    and jeq:  $j = \text{card } X$ 
    and card Y = card X
    and eq: list-of (a j0  $\cup$  ( $\bigcup_{i < \text{card } X} b (\text{enum } X i) (j0, i)$ ))
      = list-of (a j0  $\cup$  ( $\bigcup_{i < \text{card } X} b (\text{enum } Y i) (j0, i)$ ))
  have enumX:  $\bigwedge n. \llbracket n < \text{card } X \rrbracket \implies j0 \leq \text{enum } X n$ 
    using X <finite X> finite-enumerate-in-set less-imp-le-nat by blast
  have enumY:  $\bigwedge n. \llbracket n < \text{card } X \rrbracket \implies j0 \leq \text{enum } Y n$ 
    using subsetD [OF Y]
  by (metis <card Y = card X> <finite Y> finite-enumerate-in-set greaterThan-iff
less-imp-le-nat)
  have smX: strict-mono-sets  $\{.. < \text{card } X\}$  ( $\lambda i. b (\text{enum } X i) (j0, i)$ )
    and smY: strict-mono-sets  $\{.. < \text{card } X\}$  ( $\lambda i. b (\text{enum } Y i) (j0, i)$ )
    using Suc.prem1 <card Y = card X> <finite X> <finite Y> bb enumX enumY
  jeq
    by (auto simp: strict-mono-sets-def)

  have len-eq: length ms = length ns
    if (ms, ns)  $\in$  list.set (zip (map (list-of  $\circ$  ( $\lambda i. b (\text{enum } X i) (j0, i)$ ))) (list-of
 $\{.. < n\}$ ))
      (map (list-of  $\circ$  ( $\lambda i. b (\text{enum } Y i) (j0, i)$ ))) (list-of
 $\{.. < n\}$ ))
      n  $\leq$  card X
    for ms ns n
    using that
  by (induction n rule: nat.induct) (auto simp: card-b enumX enumY)
  have concat (map (list-of  $\circ$  ( $\lambda i. b (\text{enum } X i) (j0, i)$ ))) (list-of  $\{.. < \text{card } X\}$ )
    = concat (map (list-of  $\circ$  ( $\lambda i. b (\text{enum } Y i) (j0, i)$ ))) (list-of  $\{.. < \text{card } X\}$ )
    using eq

```



**by** (*simp add: sorted-list-of-set-Un less-sets-UN2 sorted-list-of-set-UN-lessThan*  
*ab enumX enumY smX smY*)  
**then have** *map-eq: map (list-of ∘ (λi. b (enum X i) (j0, i))) (list-of {..<card*  
*X})*  

$$= \text{map } (\text{list-of} \circ (\lambda i. b (\text{enum } Y \ i) (j0, i))) (\text{list-of } \{..<\text{card } X\})$$
**by** (*rule concat-injective*) (*auto simp: len-eq split: prod.split*)  
**have** *enum X i = enum Y i if i < card X for i*  
**proof** –  
**have** *Inf (b (enum X i) (j0,i)) = Inf (b (enum Y i) (j0,i))*  
**using** *iffD1 [OF map-eq-conv, OF map-eq] Suc.premis that*  
**by** (*metis (mono-tags, lifting) card-b-finite comp-apply finite-lessThan*  
*lessThan-iff set-sorted-list-of-set*)  
**moreover have** *Inf (b (enum X i) (j0,i)) ∈ (b (enum X i) (j0,i))*  
*Inf (b (enum Y i) (j0,i)) ∈ (b (enum Y i) (j0,i)) i < j0*  
**using** *Inf-nat-def1 Suc.premis b-ne enumX enumY jeq that by auto*  
**ultimately show** *?thesis*  
**by** (*metis Inf-b-less enumX enumY leI nat-less-le that*)  
**qed**  
**then show** *X = Y*  
**by** (*simp add: <card Y = card X> <finite X> <finite Y> finite-enum-ext*)  
**qed**  
**have** *BB-Suc': BB j0 (Suc j) X = usplit (λL k. BB j0 j L @ list-of (b k (j0,*  
*j))) X*  
**if** *X ∈ USigma (K j0 j) (λK. {Max (insert j0 K)<..}) for X*  
**using** *that*  
**by** (*simp add: USigma-iff K-finite less-sets-def usplit-def K-Suc BB-Suc <j ≤*  
*j0>*)  
**have** *ordertype (BB j0 (Suc j) 'K j0 (Suc j)) ?LL*  

$$= \text{ordertype}$$

$$(\text{usplit } (\lambda L k. \text{BB } j0 \ j \ L \ @ \ \text{list-of } (b \ k \ (j0, \ j))) \ ' \ \text{USigma } (K \ j0 \ j) \ (\lambda K. \ \{Max \ (insert \ j0 \ K) \ <..\})) \ ?LL$$
**by** (*simp add: BB-Suc' K-Suc*)  
**also have**  $\dots = \omega * \text{ordertype } (BB \ j0 \ j \ ' \ K \ j0 \ j) \ ?LL$   
**proof** (*intro ordertype-append-image-IJ*)  
**fix** *L k*  
**assume** *L ∈ K j0 j and k ∈ {Max (insert j0 L)<..}*  
**then have** *j0 < k and L: ∧a. a ∈ L ⇒ a < k*  
**by** (*simp-all add: K-finite*)  
**then show** *BB j0 j L < list-of (b k (j0, j))*  
**by** (*simp add: <L ∈ K j0 j> <j ≤ j0> K-finite less-list-of*)  
**next**  
**show** *inj-on (BB j0 j) (K j0 j)*  
**by** (*simp add: K-def inj-BB*)  
**next**  
**fix** *L*  
**assume** *L: L ∈ K j0 j*  
**then show** *L ≪ {Max (insert j0 L)<..} ∧ finite L*  
**by** (*simp add: K-finite less-sets-def*)  
**show** *ordertype ((λi. list-of (b i (j0, j))) ' {Max (insert j0 L)<..}) ?LL = ω*

```

    using L Suc.premS Suc-le-lessD otw by blast
  qed (auto simp: K-finite card-b)
  also have ... =  $\omega \uparrow \text{ord-of-nat } (\text{Suc } j)$ 
    by (simp add: oexp-mult-commute ih)
  finally show ?case .
qed

define seqs where seqs  $\equiv \lambda j0 j K. \text{list-of } (a j0) \# (\text{map } (\text{list-of } \circ (\lambda i. b (\text{enum } K i) (j0, i))) (\text{list-of } \{..<j\}))$ 

have length-seqs [simp]:  $\text{length } (\text{seqs } j0 j K) = \text{Suc } j$  for  $j0 j K$ 
  by (simp add: seqs-def)

have BB-eq-concat-seqs:  $BB j0 j K = \text{concat } (\text{seqs } j0 j K)$ 
  and seqs-ne:  $\text{seqs } j0 j K \in \text{lists } (- \{\{\}\})$ 
  if  $K: K \in \mathcal{K} j0 j$  and  $j \leq j0$  for  $K j j0$ 
proof -
  have  $j0: \bigwedge i. i < \text{card } K \implies j0 \leq \text{enum } K i$  and  $le-j0: \text{card } K \leq j0$ 
    using finite-enumerate-in-set that unfolding K-def nsets-def by fastforce+
  show  $BB j0 j K = \text{concat } (\text{seqs } j0 j K)$ 
    using that unfolding BB-def K-def nsets-def seqs-def
    by (fastforce simp:  $j0$  ab bb less-sets-UN2 sorted-list-of-set-Un
      strict-mono-sets-def sorted-list-of-set-UN-lessThan)
  have  $b (\text{enum } K i) (j0, i) \neq \{\}$  if  $i < \text{card } K$  for  $i$ 
    using  $j0 le-j0$  less-le-trans that by simp
  moreover have  $\text{card } K = j$ 
    using K K-card by blast
  ultimately show  $\text{seqs } j0 j K \in \text{lists } (- \{\{\}\})$ 
    by (clarsimp simp: seqs-def) (metis card-b-finite sorted-list-of-set-eq-Nil-iff)
qed

have BB-decomp:  $\exists cs. BB j0 j K = \text{concat } cs \wedge cs \in \text{lists } (- \{\{\}\})$ 
  if  $K: K \in \mathcal{K} j0 j$  and  $j \leq j0$  for  $K j j0$ 
  using BB-eq-concat-seqs seqs-ne K that(2) by blast

have a-subset-M:  $a k \subseteq M k$  for  $k$ 
  apply (clarsimp simp: a-def M-def DF-simps F-def Let-def split: prod.split-asm)
  by (metis (no-types) fst-conv fst-grab-subset nst-subset snd-conv snd-grab-subset
    subsetD)
have ba-Suc:  $b k (j, i) \ll a (\text{Suc } k)$  if  $i < j j \leq k$  for  $i j k$ 
  by (meson a-subset-M bM less-sets-weaken2 nat-less-le that)
have ba:  $b k (j, i) \ll a r$  if  $i < j j \leq k k < r$  for  $i j k r$ 
  by (metis Suc-lessI a-ne aa ba-Suc less-sets-trans that)

have disjnt-ba:  $\text{disjnt } (b k (j, i)) (a r)$  if  $i < j j \leq k$  for  $i j k r$ 
  by (meson ab ba disjnt-sym less-sets-imp-disjnt not-le that)

have bb-disjnt:  $\text{disjnt } (b k (j, i)) (b l (r, q))$ 
  if  $q < r i < j j \leq k r \leq l j < r$  for  $i j q r k l$ 

```

```

proof (cases k=l)
  case True
    with that show ?thesis
      by (force simp: pair-less-def IJ-def intro: bb-same less-sets-imp-disjnt)
  next
    case False
      with that show ?thesis
        by (metis bb less-sets-imp-disjnt disjnt-sym nat-neq-iff)
qed

have sum-card-b:  $(\sum i < j. \text{card } (b \text{ (enum } K \ i) \ (j0, \ i))) = \text{enum } (d \ j0) \ j - \text{enum } (d \ j0) \ 0$ 
  if  $K: K \subseteq \{j0 < ..\}$  finite  $K$   $\text{card } K \geq j0$  and  $j \leq j0$  for  $j0 \ j \ K$ 
  using  $\langle j \leq j0 \rangle$ 
proof (induction j)
  case 0
    then show ?case
      by auto
  next
    case (Suc j)
      then have  $j < \text{card } K$ 
        using that(3) by linarith
      have dis:  $\text{disjnt } (b \text{ (enum } K \ j) \ (j0, \ j)) \ (\bigcup i < j. b \text{ (enum } K \ i) \ (j0, \ i))$ 
        unfolding disjoint-UN-iff
      by (meson Suc.premis b-disjoint-less disjnt-def disjnt-sym lessThan-iff less-Suc-eq that)
      have j0-less:  $j0 < \text{enum } K \ j$ 
        using  $K \ \langle j < \text{card } K \rangle$  by (force simp: finite-enumerate-in-set)
      have  $(\sum i < \text{Suc } j. \text{card } (b \text{ (enum } K \ i) \ (j0, \ i))) = \text{card } (b \text{ (enum } K \ j) \ (j0, \ j)) + (\sum i < j. \text{card } (b \text{ (enum } K \ i) \ (j0, \ i)))$ 
        by (simp add: lessThan-Suc card-Un-disjnt [OF - - dis])
      also have  $\dots = \text{card } (b \text{ (enum } K \ j) \ (j0, \ j)) + \text{enum } (d \ j0) \ j - \text{enum } (d \ j0) \ 0$ 
        using  $\langle \text{Suc } j \leq j0 \rangle$  by (simp add: Suc.IH split: nat-diff-split)
      also have  $\dots = \text{enum } (d \ j0) \ (\text{Suc } j) - \text{enum } (d \ j0) \ 0$ 
        using j0-less Suc.premis card-b less-or-eq-imp-le by force
      finally show ?case .
qed

have card-UN-b:  $\text{card } (\bigcup i < j. b \text{ (enum } K \ i) \ (j0, \ i)) = \text{enum } (d \ j0) \ j - \text{enum } (d \ j0) \ 0$ 
  if  $K: K \subseteq \{j0 < ..\}$  finite  $K$   $\text{card } K \geq j0$  and  $j \leq j0$  for  $j0 \ j \ K$ 
  using that by (simp add: card-UN-disjnt sum-card-b b-disjoint)

have len-BB:  $\text{length } (BB \ j \ j \ K) = \text{enum } (d \ j) \ j$ 
  if  $K: K \in \mathcal{K} \ j \ j$  and  $j \leq j$  for  $j \ K$ 
proof -
  have dis-ab:  $\bigwedge i. i < j \implies \text{disjnt } (a \ j) \ (b \text{ (enum } K \ i) \ (j, \ i))$ 
    using  $K \ \mathcal{K}\text{-card } \mathcal{K}\text{-enum } ab \ \text{less-sets-imp-disjnt } \text{nat-less-le}$  by blast
  show ?thesis

```

```

    using K unfolding BB-def K-def nsets-def
    by (simp add: card-UN-b card-Un-disjnt dis-ab card-a cInf-le-finite finite-enumerate-in-set
enum-0-eq-Inf-finite)
qed

have d k << d (Suc k) for k
  by (metis aM a-ne d-eq da less-sets-fst-grab less-sets-trans less-sets-weaken2
next-subset)
then have dd: d k' << d k if k' < k for k' k
  by (meson UNIV-I d-ne less-sets-imp-strict-mono-sets strict-mono-sets-def that)

show thesis
proof
  show (⋃ (range XX)) ⊆ WW
    by (auto simp: XX-def BB-def WW-def)
  show ordertype (⋃ (range XX)) (?LL) = ω ↑ ω
    using otwj by (simp add: XX-def ordertype-ωω)
next
fix U
assume U: U ∈ [⋃ (range XX)]2
then obtain x y where Ueq: U = {x,y} and len-xy: length x ≤ length y
  by (auto simp: lenlex-nsets-2-eq lenlex-length)

  show ∃ l. Form l U ∧ (0 < l ⟶ [enum N l] < inter-scheme l U ∧ list.set
(inter-scheme l U) ⊆ N)
  proof (cases length x = length y)
    case True
    then show ?thesis
      using Form.intros(1) U Ueq by fastforce
  next
  case False
  then have xy: length x < length y
    using len-xy by auto
  obtain j r K L where K: K ∈ K j j and xeq: x = BB j j K and ne: BB j j
K ≠ BB r r L
    and L: L ∈ K r r and yeq: y = BB r r L
    using U by (auto simp: Ueq XX-def)
  then have length x = enum (d j) j length y = enum (d r) r
    by (auto simp: len-BB)
  then have j < r
    using xy dd
    by (metis card-d finite-enumerate-in-set finite-d lessI less-asymp less-setsD
linorder-neqE-nat)
  then have aj-ar: a j << a r
    using aa by auto
  have Ksub: K ⊆ {j<..} and finite K card K ≥ j
    using K by (auto simp: K-def nsets-def)
  have Lsub: L ⊆ {r<..} and finite L card L ≥ r
    using L by (auto simp: K-def nsets-def)

```

```

have enumK: enum K i > j if i < j for i
  using K K-card K-enum that by blast
have enumL: enum L i > r if i < r for i
  using L K-card K-enum that by blast
have list.set (acc-lengths w (seqs j0 j K)) ⊆ (+) w ‘ d j0
  if K: K ⊆ {j0<..} finite K card K ≥ j0 and j ≤ j0 for j0 j K w
  using ⟨j ≤ j0⟩
proof (induction j arbitrary: w)
  case 0
  then show ?case
    by (simp add: seqs-def Inf-nat-def1 card-a)
next
  case (Suc j)
  let ?db = ∏ (d j0) + ((∑ i<j. card (b (enum K i) (j0,i))) + card (b (enum
K j) (j0,j)))
  have j0 < enum K j
    by (meson Suc.premS Suc-le-lessD finite-enumerate-in-set greaterThan-iff
le-trans subsetD K)
  then have enum (d j0) j ≥ ∏ (d j0)
  using Suc.premS card-d by (simp add: cInf-le-finite finite-enumerate-in-set)
  then have ?db = enum (d j0) (Suc j)
  using Suc.premS that
  by (simp add: cInf-le-finite finite-enumerate-in-set sum-card-b card-b
enum-d-0 ⟨j0 < enum K j⟩ less-or-eq-imp-le)
  then have ?db ∈ d j0
  using Suc.premS finite-enumerate-in-set by (auto simp: finite-enumerate-in-set)
  moreover have list.set (acc-lengths w (seqs j0 j K)) ⊆ (+) w ‘ d j0
  by (simp add: Suc Suc-leD)
  then have list.set (acc-lengths (w + ∏ (d j0))
    (map (list-of ∘ (λi. b (enum K i) (j0,i))) (list-of {..<j})))
    ⊆ (+) w ‘ d j0
  by (simp add: seqs-def card-a subset-insertI)
  ultimately show ?case
  by (simp add: seqs-def acc-lengths-append image-iff Inf-nat-def1
sum-sorted-list-of-set-map card-a)
qed
then have acc-lengths-subset-d: list.set (acc-lengths 0 (seqs j0 j K)) ⊆ d j0
  if K: K ⊆ {j0<..} finite K card K ≥ j0 and j ≤ j0 for j0 j K
  by (metis image-add-0 that)

have strict-sorted x strict-sorted y
  by (auto simp: xeq yeq BB-def)
have disjnt-xy: disjnt (list.set x) (list.set y)
proof –
  have disjnt (a j) (a r)
    using ⟨j < r⟩ aa less-sets-imp-disjnt by blast
  moreover have disjnt (b (enum K i) (j,i)) (a r) if i < j for i
    by (simp add: disjnt-ba enumK less-imp-le-nat that)
  moreover have disjnt (a j) (b (enum L q) (r,q)) if q < r for q

```

```

    by (meson disjnt-ba disjnt-sym enumL less-imp-le-nat that)
  moreover have disjnt (b (enum K i) (j,i)) (b (enum L q) (r,q)) if i < j q
< r for i q
  by (meson ⟨j < r⟩ bb-disjnt enumK enumL less-imp-le that)
ultimately show ?thesis
  by (simp add: xeq yeq BB-def)
qed
have ∃ us vs. merge (seqs j j K) (seqs r r L) us vs
proof (rule merge-exists)
  show strict-sorted (concat (seqs j j K))
    using BB-eq-concat-seqs K ⟨strict-sorted x⟩ xeq by auto
  show strict-sorted (concat (seqs r r L))
    using BB-eq-concat-seqs L ⟨strict-sorted y⟩ yeq by auto
  show seqs j j K ∈ lists (− {[]}) seqs r r L ∈ lists (− {[]})
    by (auto simp: K L seqs-ne)
  show hd (seqs j j K) < hd (seqs r r L)
    by (simp add: aj-ar less-sets-imp-list-less seqs-def)
  show seqs j j K ≠ [] seqs r r L ≠ []
    using seqs-def by blast+
  have less-bb: b (enum K i) (j,i) ≪ b (enum L p) (r, p)
    if ¬ b (enum L p) (r, p) ≪ b (enum K i) (j,i) and i < j p < r
    for i p
    by (metis IJ-iff ⟨j < r⟩ bb bb-same enumK enumL less-imp-le-nat
linorder-neqE-nat pair-lessI1 that)
  show u < v ∨ v < u
    if u ∈ list.set (seqs j j K) and v ∈ list.set (seqs r r L) for u v
    using that enumK enumL unfolding seqs-def
    apply (auto simp: seqs-def aj-ar intro!: less-bb less-sets-imp-list-less)
    apply (meson ab ba less-imp-le-nat not-le)+
    done
qed
then obtain uus vvs where merge: merge (seqs j j K) (seqs r r L) uus vvs
  by metis
then have uus ≠ []
  using merge-length1-gt-0 by (auto simp: seqs-def)
then obtain u1 us where us: u1 # us = uus
  by (metis neq-Nil-conv)
define ku where ku ≡ length (u1 # us)
define ps where ps ≡ acc-lengths 0 (u1 # us)
have us-ne: u1 # us ∈ lists (− {[]})
  using merge-length1-nonempty seqs-ne us merge us K by auto
have xu-eq: x = concat (u1 # us)
  using BB-eq-concat-seqs K merge merge-preserves us xeq by auto
then have strict-sorted u1
  using ⟨strict-sorted x⟩ strict-sorted-append-iff by auto
have u-sub: list.set ps ⊆ list.set (acc-lengths 0 (seqs j j K))
  using acc-lengths-merge1 merge ps-def us by blast
have vvs ≠ []
  using merge BB-eq-concat-seqs L merge-preserves xy yeq by auto

```

```

then obtain  $v1\ vs$  where  $vs: v1\ \#vs = vvs$ 
  by (metis neq-Nil-conv)
define  $kv$  where  $kv \equiv length\ (v1\ \#vs)$ 
define  $qs$  where  $qs \equiv acc-lengths\ 0\ (v1\ \#vs)$ 
have  $vs-ne: v1\ \#vs \in lists\ (-\ \{\}\}$ 
  using  $L\ merge\ merge-length2-nonempty\ seqs-ne\ vs$  by auto
have  $yv-eq: y = concat\ (v1\ \#vs)$ 
  using  $BB-eq-concat-seqs\ L\ merge\ merge-preserves\ vs\ yeq$  by auto
then have strict-sorted  $v1$ 
  using  $\langle strict-sorted\ y \rangle\ strict-sorted-append-iff$  by auto
have  $v-sub: list.set\ qs \subseteq list.set\ (acc-lengths\ 0\ (seqs\ r\ r\ L))$ 
  using  $acc-lengths-merge2\ merge\ qs-def\ vs$  by blast

have  $ss-concat-jj: strict-sorted\ (concat\ (seqs\ j\ j\ K))$ 
  using  $BB-eq-concat-seqs\ K\ \langle strict-sorted\ x \rangle\ xeq$  by auto
then obtain  $k: 0 < kv\ kv \leq ku\ ku \leq Suc\ kv\ kv \leq Suc\ j$ 
  using  $us\ vs\ merge-length-le\ merge-length-le-Suc\ merge-length-less2\ merge$ 
  unfolding  $ku-def\ kv-def$  by fastforce

define  $zs$  where  $zs \equiv concat\ [ps, u1, qs, v1] @ interact\ us\ vs$ 
have  $ss: strict-sorted\ zs$ 
proof –
  have  $ssp: strict-sorted\ ps$ 
    unfolding  $ps-def$  by (meson strict-sorted-acc-lengths us-ne)
  have  $ssq: strict-sorted\ qs$ 
    unfolding  $qs-def$  by (meson strict-sorted-acc-lengths vs-ne)

  have  $d\ j \ll list.set\ x$ 
    using  $da\ [of\ j]\ db\ [of\ j]\ K\ \mathcal{K}\text{-card}\ \mathcal{K}\text{-enum}\ nat\text{-less-le}$ 
    by (auto simp: xeq BB-def less-sets-Un2 less-sets-UN2)
  then have  $ac-x: acc-lengths\ 0\ (seqs\ j\ j\ K) < x$ 
    by (meson Ksub  $\langle finite\ K \rangle\ \langle j \leq card\ K \rangle\ acc-lengths-subset-d\ le-refl$ 
less-sets-imp-list-less less-sets-weaken1)
  then have  $ps < x$ 
    by (meson Ksub  $\langle d\ j \ll list.set\ x \rangle\ \langle finite\ K \rangle\ \langle j \leq card\ K \rangle\ acc-lengths-subset-d$ 
le-refl less-sets-imp-list-less less-sets-weaken1 u-sub)
  then have  $ps < u1$ 
    by (metis Nil-is-append-conv concat.simps(2) hd-append2 less-list-def xu-eq)

  have  $d\ r \ll list.set\ y$ 
    using  $da\ [of\ r]\ db\ [of\ r]\ L\ \mathcal{K}\text{-card}\ \mathcal{K}\text{-enum}\ nat\text{-less-le}$ 
    by (auto simp: yeq BB-def less-sets-Un2 less-sets-UN2)
  then have  $acc-lengths\ 0\ (seqs\ r\ r\ L) < y$ 
    by (meson Lsub  $\langle finite\ L \rangle\ \langle r \leq card\ L \rangle\ acc-lengths-subset-d\ le-refl$ 
less-sets-imp-list-less less-sets-weaken1)
  then have  $qs < y$ 
    by (metis L Lsub  $\mathcal{K}\text{-card}\ \langle d\ r \ll list.set\ y \rangle\ \langle finite\ L \rangle\ acc-lengths-subset-d$ 
less-sets-imp-list-less less-sets-weaken1 order-refl v-sub)
  then have  $qs < v1$ 

```

**by** (*metis concat.simps(2) gr-implies-not0 hd-append2 less-list-def list.size(3) xy yv-eq*)

**have** *carda-v1*:  $\text{card } (a \ r) \leq \text{length } v1$   
**using** *length-hd-merge2 [OF merge] unfolding vs [symmetric]* **by** (*simp add: seqs-def*)

**have** *ab-enumK*:  $\bigwedge i. i < j \implies a \ j \ll b \ (\text{enum } K \ i) \ (j, i)$   
**by** (*meson ab enumK le-trans less-imp-le-nat*)

**have** *ab-enumL*:  $\bigwedge q. q < r \implies a \ j \ll b \ (\text{enum } L \ q) \ (r, q)$   
**by** (*meson <j < r> ab enumL le-trans less-imp-le-nat*)

**then have** *ay*:  $a \ j \ll \text{list.set } y$   
**by** (*auto simp: yeq BB-def less-sets-Un2 less-sets-UN2 aj-ar*)

**have** *disjnt-hd-last-K-y*:  $\text{disjnt } \{\text{hd } l.. \text{last } l\} \ (\text{list.set } y)$   
**if**  $l \in \text{list.set } (\text{seqs } j \ j \ K)$  **for**  $l$   
**proof** (*clarsimp simp add: yeq BB-def disjnt-iff Ball-def, intro conjI strip*)  
**fix**  $u$   
**assume**  $u \leq \text{last } l$  **and**  $\text{hd } l \leq u$   
**with**  $l$  **consider**  $u \leq \text{last } (\text{list-of } (a \ j)) \ \text{hd } (\text{list-of } (a \ j)) \leq u$   
|  $i$  **where**  $i < j \ u \leq \text{last } (\text{list-of } (b \ (\text{enum } K \ i) \ (j, i))) \ \text{hd } (\text{list-of } (b \ (\text{enum } K \ i) \ (j, i))) \leq u$   
**by** (*force simp: seqs-def*)  
**note** *l-cases = this*  
**then show**  $u \notin a \ r$   
**proof** *cases*  
**case 1**  
**then show** *?thesis*  
**by** (*metis a-ne aj-ar finite-a last-in-set leD less-setsD set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff*)  
**next**  
**case 2**  
**then show** *?thesis*  
**by** (*metis enumK ab ba Inf-nat-def1 b-ne card-b-finite hd-b last-in-set less-asym less-setsD not-le set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff*)  
**qed**  
**fix**  $q$   
**assume**  $q < r$   
**show**  $u \notin b \ (\text{enum } L \ q) \ (r, q)$   
**using** *l-cases*  
**proof** *cases*  
**case 1**  
**then show** *?thesis*  
**by** (*metis <q < r> a-ne ab-enumL finite-a last-in-set leD less-setsD set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff*)  
**next**  
**case 2**  
**show** *?thesis*  
**proof** (*cases enum K i = enum L q*)



```

    case True
    then show ?thesis
      using 2 bb-same [of concl: enum L q j i r q] ⟨j < r⟩ u
      by (metis JJ-iff b-ne card-b-finite enumK last-in-set leD less-imp-le-nat
less-setsD pair-lessI1 set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff)
    next
    case False
    with 2 bb enumK enumL show ?thesis
      unfolding less-sets-def
      by (metis ⟨q < r⟩ b-ne card-b-finite last-in-set leD less-imp-le-nat
list.set-sel(1) nat-neq-iff set-sorted-list-of-set sorted-list-of-set-eq-Nil-iff)
  qed
  qed
  qed

  have u1-y: list.set u1 ≪ list.set y
    using vs yv-eq L ⟨strict-sorted y⟩ merge merge-less-sets-hd merge-preserves
seqs-ne ss-concat-jj us by fastforce
  have u1-subset-seqs: list.set u1 ⊆ list.set (concat (seqs j j K))
    using merge-preserves [OF merge] us by auto

  have b k (j,i) ≪ d (Suc k) if j ≤ k i < j for k j i
    by (metis bM d-eq less-sets-fst-grab less-sets-weaken2 nxt-subset that)
  then have bd: b k (j,i) ≪ d k' if j ≤ k i < j k < k' for k k' j i
    by (metis Suc-lessI d-ne dd less-sets-trans that)

  have a k ≪ d (Suc k) for k
    by (metis aM d-eq less-sets-fst-grab less-sets-weaken2 nxt-subset)
  then have ad: a k ≪ d k' if k < k' for k k'
    by (metis Suc-lessI d-ne dd less-sets-trans that)

  have u1 < y
    by (simp add: u1-y less-sets-imp-list-less)
  have n < Inf (d r) if n: n ∈ list.set u1 for n
  proof -
    obtain l where l: l ∈ list.set (seqs j j K) and n: n ∈ list.set l
      using n u1-subset-seqs by auto
    then consider l = list-of (a j) | i where l = list-of (b (enum K i) (j,i))
i < j
      by (force simp: seqs-def)
    then show ?thesis
  proof cases
    case 1
    then show ?thesis
  by (metis Inf-nat-def1 ⟨j < r⟩ ad d-ne finite-a less-setsD n set-sorted-list-of-set)
  next
  case 2
  then have hd (list-of (b (enum K i) (j,i))) = Min (b (enum K i) (j,i))
    by (meson b-ne card-b-finite enumK hd-list-of less-imp-le-nat)

```

**also have**  $\dots \leq n$   
**using**  $2\ n$  **by** (*simp add: less-list-def disjnt-iff less-sets-def*)  
**also have**  $f8: n < \text{hd } y$   
**using** *less-setsD that u1-y*  
**by** (*metis gr-implies-not0 list.set-sel(1) list.size(3) xy*)  
**finally have**  $l < y$   
**using**  $2\ \text{disjnt-hd-last-}K\text{-}y$  [*OF l*]  
**by** (*simp add: disjnt-iff*) (*metis leI less-imp-le-nat less-list-def*  
*list.set-sel(1)*)  
**moreover have**  $\text{last } (\text{list-of } (b\ (\text{enum } K\ i)\ (j,i))) < \text{hd } (\text{list-of } (a\ r))$   
**using**  $\langle l < y \rangle L\ n$  **by** (*auto simp: 2yeq BB-eq-concat-seqs seqs-def*  
*less-list-def*)  
**then have**  $\text{enum } K\ i < r$   
**by** (*metis 2(1) a-ne ab card-b-finite empty-iff finite.emptyI finite-a*  
*last-in-set leI less-asm less-setsD list.set-sel(1) n set-sorted-list-of-set*)  
**moreover have**  $j \leq \text{enum } K\ i$   
**by** (*simp add: 2(2) enumK less-imp-le-nat*)  
**ultimately show** *?thesis*  
**using**  $2\ n\ \text{bd}$  [*of j enum K i i r*] *Inf-nat-def1 less-setsD* **by** *force*  
**qed**  
**then have**  $\text{last } u1 < \text{Inf } (d\ r)$   
**using**  $\langle u1 \neq [] \rangle\ us\text{-ne}$  **by** *auto*  
**also have**  $\dots \leq \text{length } v1$   
**using** *card-a carda-v1* **by** *auto*  
**finally have**  $\text{last } u1 < \text{length } v1$  .  
**then have**  $u1 < qs$   
**by** (*simp add: qs-def less-list-def*)  
  
**have** *strict-sorted* (*interact* ( $u1\ \#u1$ ) ( $v1\ \#v1$ ))  
**using**  $L\ \langle \text{strict-sorted } x \rangle\ \langle \text{strict-sorted } y \rangle$  *merge merge-interact merge-preserves*  
*seqs-ne us vs xu-eq yv-eq* **by** *auto*  
**then have** *strict-sorted* (*interact us vs*)  $v1 < \text{interact } us\ vs$   
**by** (*auto simp: strict-sorted-append-iff*)  
**moreover have**  $ps < u1\ @\ qs\ @\ v1\ @\ \text{interact } us\ vs$   
**using**  $\langle ps < u1 \rangle\ us\text{-ne}$  **unfolding** *less-list-def* **by** *auto*  
**moreover have**  $u1 < qs\ @\ v1\ @\ \text{interact } us\ vs$   
**by** (*metis*  $\langle u1 < qs \rangle\ \langle v1 \neq [] \rangle\ \text{acc-lengths-eq-}Nil\text{-iff hd-append less-list-def}$   
*qs-def vs*)  
**moreover have**  $qs < v1\ @\ \text{interact } us\ vs$   
**using**  $\langle qs < v1 \rangle\ us\text{-ne}$   $\langle \text{last } u1 < \text{length } v1 \rangle\ vs\text{-ne}$  **by** (*auto simp:*  
*less-list-def*)  
**ultimately show** *?thesis*  
**by** (*simp add: zs-def strict-sorted-append-iff ssp ssq*  $\langle \text{strict-sorted } u1 \rangle$   
 $\langle \text{strict-sorted } v1 \rangle$ )  
**qed**  
**have** *ps-subset-d: list.set ps*  $\subseteq d\ j$   
**using**  $K\ Ksub\ \mathcal{K}\text{-card}$   $\langle \text{finite } K \rangle\ \text{acc-lengths-subset-d } u\text{-sub}$  **by** *blast*  
**have** *ps-less-u1: ps*  $< u1$

```

    by (metis append.assoc concat.simps(2) ss strict-sorted-append-iff zs-def)
  have qs-subset-d: list.set qs  $\subseteq$  d r
    using L Lsub K-card ⟨finite L⟩ acc-lengths-subset-d v-sub by blast
  have qs-less-v1: qs < v1
    by (metis append.assoc concat.simps(2) ss strict-sorted-append-iff zs-def)
  have FB: Form-Body ku kv x y zs
    unfolding Form-Body.simps ku-def kv-def
    using ps-def qs-def ss us-ne vs-ne xu-eq xy yv-eq zs-def by blast
  then have zs = (inter-scheme ((ku+kv) - Suc 0) {x,y})
    by (simp add: Form-Body-imp-inter-scheme k)
  obtain l where l ≤ 2 * (Suc j) and l: Form l U and zs-eq-interact: zs =
inter-scheme l {x,y}
  proof
    show ku+kv-1 ≤ 2 * (Suc j)
      using k by auto
    show Form (ku+kv-1) U
    proof (cases ku=kv)
      case True
        then show ?thesis
          using FB Form.simps Ueq ⟨0 < kv⟩ by (auto simp: mult-2)
      next
        case False
          then have ku = Suc kv
            using k by auto
          then show ?thesis
            using FB Form.simps Ueq ⟨0 < kv⟩ by auto
    qed
    show zs = inter-scheme (ku + kv - 1) {x, y}
      using Form-Body-imp-inter-scheme by (simp add: FB k)
    qed
  then have enum N l ≤ enum N (Suc (2 * Suc j))
    by (simp add: assms less-imp-le-nat)
  also have ... < Min (d j)
  by (smt (verit, best) Min-gr-iff d-eq d-ne finite-d fst-grab-subset greaterThan-iff
in-mono le-inf-iff nxt-def)
  finally have ls: {enum N l}  $\ll$  d j
    by simp
  have l > 0
    by (metis l False Form-0-cases-raw Set.doubleton-eq-iff Ueq grOI)
  show ?thesis
    unfolding Ueq
  proof (intro exI conjI impI)
    have zs-subset: list.set zs  $\subseteq$  list.set (acc-lengths 0 (seqs j j K))  $\cup$  list.set
(acc-lengths 0 (seqs r r L))  $\cup$  list.set x  $\cup$  list.set y
      using u-sub v-sub by (auto simp: zs-def xu-eq yv-eq)
    also have ...  $\subseteq$  N
  proof (simp, intro conjI)
    show list.set (acc-lengths 0 (seqs j j K))  $\subseteq$  N
      using d-subset-N Ksub ⟨finite K⟩ ⟨j ≤ card K⟩ acc-lengths-subset-d by

```

```

blast
  show list.set (acc-lengths 0 (seqs r r L))  $\subseteq$  N
    using d-subset-N Lsub  $\langle$ finite L $\rangle$   $\langle$ r  $\leq$  card L $\rangle$  acc-lengths-subset-d by
blast
  show list.set x  $\subseteq$  N list.set y  $\subseteq$  N
    by (simp-all add: xeq yeq BB-def a-subset-N UN-least b-subset-N)
qed
finally show list.set (inter-scheme l {x, y})  $\subseteq$  N
  using zs-eq-interact by blast
have [enum N l] < ps
  using ps-subset-d ls
  by (metis empty-set less-sets-imp-list-less less-sets-weaken2 list.simps(15))
then show [enum N l] < inter-scheme l {x, y}
  by (simp add: zs-def less-list-def ps-def flip: zs-eq-interact)
qed (use Ueq l in blast)
qed
qed
qed

```

### 3.12 The main partition theorem for $\omega \uparrow \omega$

**definition** *iso-ll* where  $iso-ll\ A\ B \equiv iso\ (lenlex\ less-than \cap (A \times A))\ (lenlex\ less-than \cap (B \times B))$

**corollary** *ordertype-eq-ordertype-iso-ll*:

assumes  $Field\ (Restr\ (lenlex\ less-than)\ A) = A\ Field\ (Restr\ (lenlex\ less-than)\ B) = B$

shows  $(ordertype\ A\ (lenlex\ less-than) = ordertype\ B\ (lenlex\ less-than)) \iff (\exists f. iso-ll\ A\ B\ f)$

**proof** –

have  $total-on\ A\ (lenlex\ less-than) \wedge total-on\ B\ (lenlex\ less-than)$

by (*meson UNIV-I total-lenlex total-on-def total-on-less-than*)

then show *?thesis*

by (*simp add: assms wf-lenlex lenlex-transI iso-ll-def ordertype-eq-ordertype-iso-Restr*)

qed

**theorem** *partition- $\omega\omega$ -aux*:

assumes  $\alpha \in elts\ \omega$

shows  $partn-lst\ (lenlex\ less-than)\ WW\ [\omega \uparrow \omega, \alpha]\ 2\ (is\ partn-lst\ ?R\ WW\ [\omega \uparrow \omega, \alpha]\ 2)$

**proof** (*cases*  $\alpha \leq 1$ )

case *True*

then show *?thesis*

using *strict-sorted-into-WW unfolding WW-def* by (*auto intro!: partn-lst-triv1 [where  $i=1$ ]*)

next

case *False*

obtain *m* where  $m: \alpha = ord-of-nat\ m$

using *assms elts- $\omega$*  by *auto*

```

then have m>1
  using False by auto
show ?thesis
  unfolding partn-lst-def
proof clarsimp
  fix f
  assume f: f ∈ [WW]2 → {..Suc (Suc 0)}
  let ?P0 = ∃ X ⊆ WW. ordertype X ?R = ω↑ω ∧ f ' [X]2 ⊆ {0}
  let ?P1 = ∃ M ⊆ WW. ordertype M ?R = α ∧ f ' [M]2 ⊆ {1}
  have †: ?P0 ∨ ?P1
  proof (rule disjCI)
    assume not1: ¬ ?P1
    have ∃ W'. ordertype W' ?R = ω↑n ∧ f ' [W']2 ⊆ {0} ∧ W' ⊆ WW-seg
      (n*m) for n::nat
    proof -
      have fnm: f ∈ [WW-seg (n*m)]2 → {..Suc (Suc 0)}
      using f WW-seg-subset-WW [of n*m] by (meson in-mono nsets-Pi-contr)
      have *: partn-lst ?R (WW-seg (n*m)) [ω↑n, ord-of-nat m] 2
        using ordertype-WW-seg [of n*m]
        by (simp add: partn-lst-VWF-imp-partn-lst [OF Theorem-3-2])
      show ?thesis
        using partn-lst-E [OF * fnm, simplified]
        by (metis One-nat-def WW-seg-subset-WW less-2-cases m not1 nth-Cons-0
          nth-Cons-Suc numeral-2-eq-2 subset-trans)
    qed
    then obtain W':: nat ⇒ nat list set
      where otW': ∧n. ordertype (W' n) ?R = ω↑n
      and f-W': ∧n. f ' [W' n]2 ⊆ {0}
      and seg-W': ∧n. W' n ⊆ WW-seg (n*m)
      by metis
    define WW' where WW' ≡ (∪ n. W' n)
    have WW' ⊆ WW
      using seg-W' WW-seg-subset-WW by (force simp: WW'-def)
    with f have f': f ∈ [WW]2 → {..Suc (Suc 0)}
      using nsets-mono by fastforce
    have ot': ordertype WW' ?R = ω↑ω
    proof (rule antisym)
      have ordertype WW' ?R ≤ ordertype WW ?R
        by (simp add: ‹WW' ⊆ WW› lenlex-transI ordertype-mono wf-lenlex)
      with ordertype-WW
      show ordertype WW' ?R ≤ ω ↑ ω
        by simp
      have ω ↑ n ≤ ordertype (∪ (range W')) ?R for n::nat
        using oexp-Limit ordertype-ω otW' by auto
      then show ω ↑ ω ≤ ordertype WW' ?R
        by (auto simp: elts-ω oexp-Limit ZFC-in-HOL.SUP-le-iff WW'-def)
    qed
    have FR-WW: Field (Restr (lenlex less-than) WW) = WW
      by (simp add: Limit-omega-oexp Limit-ordertype-imp-Field-Restr order-

```

```

type-WW)
  have FR-WW': Field (Restr (lenlex less-than) WW') = WW'
    by (simp add: Limit-omega-oexp Limit-ordertype-imp-Field-Restr ot')
  have FR-W: Field (Restr (lenlex less-than) (WW-seg n)) = WW-seg n if
n>0 for n
  by (simp add: Limit-omega-oexp ordertype-WW-seg that Limit-ordertype-imp-Field-Restr)
  have FR-W': Field (Restr (lenlex less-than) (W' n)) = W' n if n>0 for n
    by (simp add: Limit-omega-oexp otW' that Limit-ordertype-imp-Field-Restr)
  have  $\exists h. \text{iso-ll } (WW\text{-seg } n) (W' n) h$  if n>0 for n
  proof (subst ordertype-eq-ordertype-iso-ll [symmetric])
    show ordertype (WW-seg n) (lenlex less-than) = ordertype (W' n) (lenlex
less-than)
      by (simp add: ordertype-WW-seg otW')
    qed (auto simp: FR-W FR-W' that)
  then obtain h-seg where h-seg:  $\bigwedge n. n > 0 \implies \text{iso-ll } (WW\text{-seg } n) (W' n)$ 
(h-seg n)
    by metis
  define h where h  $\equiv \lambda l. \text{if } l = [] \text{ then } [] \text{ else } h\text{-seg } (\text{length } l) l$ 

  have bij-h-seg:  $\bigwedge n. n > 0 \implies \text{bij-betw } (h\text{-seg } n) (WW\text{-seg } n) (W' n)$ 
    using h-seg by (simp add: iso-ll-def iso-iff2 FR-W FR-W')
  have len-h-seg:  $\text{length } (h\text{-seg } (\text{length } l) l) = \text{length } l * m$ 
    if  $\text{length } l > 0 \ l \in WW$  for l
    using bij-betwE [OF bij-h-seg] seg-W' that by (simp add: WW-seg-def
subset-iff)
  have hlen:  $\text{length } (h x) = \text{length } (h y) \iff \text{length } x = \text{length } y$  if  $x \in WW \ y$ 
 $\in WW$  for x y
    using that <1 < m> h-def len-h-seg by force

  have h: iso-ll WW WW' h
    unfolding iso-ll-def iso-iff2 FR-WW FR-WW'
  proof (intro conjI strip)
    have W'-ne:  $W' n \neq \{\}$  for n
      using otW' [of n] by auto
    then have  $[] \in WW'$ 
      using seg-W' [of 0] by (auto simp: WW'-def WW-seg-def)
    let ?g =  $\lambda l. \text{if } l = [] \text{ then } l \text{ else } \text{inv-into } (WW\text{-seg } (\text{length } l \text{ div } m)) (h\text{-seg}$ 
(length l div m)) l
      have h-seg-iff:  $\bigwedge n \ a \ b. [a \in WW\text{-seg } n; b \in WW\text{-seg } n; n > 0] \implies$ 
 $(a, b) \in \text{lenlex less-than} \iff$ 
 $(h\text{-seg } n \ a, h\text{-seg } n \ b) \in \text{lenlex less-than} \wedge h\text{-seg } n \ a \in W' n$ 
 $\wedge h\text{-seg } n \ b \in W' n$ 
        using h-seg by (auto simp: iso-ll-def iso-iff2 FR-W FR-W')

  show bij-betw h WW WW'
    unfolding bij-betw-iff-bijections
  proof (intro exI conjI ballI)
    fix l
    assume l  $\in WW$ 

```

```

then have  $l \in WW\text{-seg } (\text{length } l)$ 
  by (simp add: WW-seg-def)
have  $h \ l \in W' (\text{length } l)$ 
proof (cases l=[])
  case True
    with seg-W' [of 0] W'-ne show ?thesis
      by (auto simp: WW-seg-def h-def)
  next
    case False
      then show ?thesis
        using bij-betwE bij-h-seg h-def l by fastforce
qed
show  $h \ l \in WW'$ 
  using WW'-def ⟨h l ∈ W' (length l)⟩ by blast
show  $?g (h \ l) = l$ 
proof (cases l=[])
  case False
    then have  $\text{length } l > 0$ 
      by auto
    then have  $h\text{-seg } (\text{length } l) \ l \neq []$ 
      using  $\langle 1 < m \rangle \langle l \in WW \rangle \text{len-h-seg}$  by fastforce
      moreover have  $\text{bij-betw } (h\text{-seg } (\text{length } l)) (WW\text{-seg } (\text{length } l)) (W'$ 
(length l)
        using  $\langle 0 < \text{length } l \rangle \text{bij-h-seg}$  by presburger
        ultimately show ?thesis
          using  $\langle l \in WW \rangle \text{bij-betw-inv-into-left } h\text{-def } l \text{len-h-seg}$  by fastforce
qed (auto simp: h-def)
next
fix  $l$ 
assume  $l \in WW'$ 
then have  $l \in W' (\text{length } l \text{ div } m)$ 
  using WW-seg-def ⟨1 < m⟩ seg-W' by (fastforce simp: WW'-def)
show  $?g \ l \in WW$ 
proof (cases l=[])
  case False
    then have  $l \notin W' \ 0$ 
      using WW-seg-def seg-W' by fastforce
      with  $l$  have  $\text{inv-into } (WW\text{-seg } (\text{length } l \text{ div } m)) (h\text{-seg } (\text{length } l \text{ div } m))$ 
 $l \in WW\text{-seg } (\text{length } l \text{ div } m)$ 
      by (metis Nat.neq0-conv bij-betwE bij-betw-inv-into bij-h-seg)
      then show ?thesis
        using False WW-seg-subset-WW by auto
qed (auto simp: WW-def)

show  $h (?g \ l) = l$ 
proof (cases l=[])
  case False
    then have  $0 < \text{length } l \text{ div } m$ 
      using WW-seg-def l seg-W' by fastforce

```

**then have**  $inv\text{-}into$  ( $WW\text{-}seg$  ( $length$   $l$   $div$   $m$ )) ( $h\text{-}seg$  ( $length$   $l$   $div$   $m$ ))  $l$   
 $\in WW\text{-}seg$  ( $length$   $l$   $div$   $m$ )  
**by** ( $metis$   $bij\text{-}betw\text{-}imp\text{-}surj\text{-}on$   $bij\text{-}h\text{-}seg$   $inv\text{-}into\text{-}into$   $l$ )  
**then show**  $?thesis$   
**using**  $bij\text{-}h\text{-}seg$  [ $of$   $length$   $l$   $div$   $m$ ]  $WW\text{-}seg\text{-}def$   $\langle 0 < length$   $l$   $div$   $m \rangle$   
 $bij\text{-}betw\text{-}inv\text{-}into\text{-}right$   $l$   
**by** ( $fastforce$   $simp$ :  $h\text{-}def$ )  
**qed** ( $auto$   $simp$ :  $h\text{-}def$ )  
**qed**  
**fix**  $a$   $b$   
**assume**  $a \in WW$   $b \in WW$   
**show**  $(a, b) \in Restr$  ( $lenlex$   $less\text{-}than$ )  $WW \longleftrightarrow (h$   $a, h$   $b) \in Restr$  ( $lenlex$   
 $less\text{-}than$ )  $WW'$   
**(is**  $?lhs = ?rhs$ )  
**proof**  
**assume**  $L$ :  $?lhs$   
**then consider**  $length$   $a < length$   $b$  |  $length$   $a = length$   $b$  ( $a, b) \in lex$   
 $less\text{-}than$   
**by** ( $auto$   $simp$ :  $lenlex\text{-}conv$ )  
**then show**  $?rhs$   
**proof cases**  
**case 1**  
**then have**  $length$  ( $h$   $a$ )  $<$   $length$  ( $h$   $b$ )  
**using**  $\langle 1 < m \rangle$   $\langle a \in WW \rangle$   $\langle b \in WW \rangle$   $h\text{-}def$   $len\text{-}h\text{-}seg$  **by**  $auto$   
**then have** ( $h$   $a, h$   $b$ )  $\in lenlex$   $less\text{-}than$   
**by** ( $auto$   $simp$ :  $lenlex\text{-}conv$ )  
**then show**  $?thesis$   
**using**  $\langle a \in WW \rangle$   $\langle b \in WW \rangle$   $\langle bij\text{-}betw$   $h$   $WW$   $WW' \rangle$   $bij\text{-}betwE$  **by**  
 $fastforce$   
**next**  
**case 2**  
**then have**  $ab$ :  $a \in WW\text{-}seg$  ( $length$   $a$ )  $b \in WW\text{-}seg$  ( $length$   $a$ )  
**using**  $\langle a \in WW \rangle$   $\langle b \in WW \rangle$  **by** ( $auto$   $simp$ :  $WW\text{-}seg\text{-}def$ )  
**have**  $length$  ( $h$   $a$ )  $=$   $length$  ( $h$   $b$ )  
**using**  $2$   $\langle a \in WW \rangle$   $\langle b \in WW \rangle$   $h\text{-}def$   $len\text{-}h\text{-}seg$  **by**  $force$   
**moreover have** ( $a, b$ )  $\in lenlex$   $less\text{-}than$   
**using**  $L$  **by**  $blast$   
**then have** ( $h\text{-}seg$  ( $length$   $a$ )  $a, h\text{-}seg$  ( $length$   $a$ )  $b$ )  $\in lenlex$   $less\text{-}than$   
**using**  $2$   $ab$   $h\text{-}seg\text{-}iff$  **by**  $blast$   
**ultimately show**  $?thesis$   
**using**  $2$   $\langle a \in WW \rangle$   $\langle b \in WW \rangle$   $\langle bij\text{-}betw$   $h$   $WW$   $WW' \rangle$   $bij\text{-}betwE$   $h\text{-}def$   
**by**  $fastforce$   
**qed**  
**next**  
**assume**  $R$ :  $?rhs$   
**then have**  $R'$ : ( $h$   $a, h$   $b$ )  $\in lenlex$   $less\text{-}than$   
**by**  $blast$   
**then consider**  $length$   $a < length$   $b$   
|  $length$   $a = length$   $b$  ( $h$   $a, h$   $b$ )  $\in lex$   $less\text{-}than$



```

    using ⟨a ∈ WW⟩ ⟨b ∈ WW⟩ ⟨m > 1⟩
    by (auto simp: lenlex-conv h-def len-h-seg split: if-split-asm)
  then show ?lhs
  proof cases
    case 1
    then show ?thesis
      using omega-sum-less-iff ⟨a ∈ WW⟩ ⟨b ∈ WW⟩ by auto
    next
    case 2
    then have ab: a ∈ WW-seg (length a) b ∈ WW-seg (length a)
      using ⟨a ∈ WW⟩ ⟨b ∈ WW⟩ by (auto simp: WW-seg-def)
    then have (a, b) ∈ lenlex less-than
      using bij-betwE [OF bij-h-seg] ⟨a ∈ WW⟩ ⟨b ∈ WW⟩ R' 2
      by (simp add: h-def h-seg-iff split: if-split-asm)
    then show ?thesis
      using ⟨a ∈ WW⟩ ⟨b ∈ WW⟩ by blast
  qed
qed
qed

let ?fh = f ∘ image h
have bij-betw h WW WW'
  using h unfolding iso-ll-def iso-iff2 by (fastforce simp: FR-WW FR-WW')
moreover have {..Suc (Suc 0)} = {0,1}
  by auto
ultimately have fh: ?fh ∈ [WW]2 → {0,1}
unfolding Pi-iff using bij-betwE f' bij-betw-nsets by (metis PiE comp-apply)
have f{x,y} = 0 if x ∈ WW' y ∈ WW' length x = length y x ≠ y for x y
proof -
  obtain p q where x ∈ W' p and y ∈ W' q
    using WW'-def ⟨x ∈ WW'⟩ ⟨y ∈ WW'⟩ by blast
  then obtain n where {x,y} ∈ [W' n]2
    using seg-W' ⟨1 < m⟩ ⟨length x = length y⟩ ⟨x ≠ y⟩
    by (auto simp: WW'-def WW-seg-def subset-iff)
  then show f{x,y} = 0
    using f-W' by blast
qed
then have fh-eq-0-eqlen: ?fh{x,y} = 0 if x ∈ WW y ∈ WW length x = length
y x ≠ y for x y
  using ⟨bij-betw h WW WW'⟩ that hlen by (simp add: bij-betw-iff-bijections)
metis
have m-f-0: ∃ x ∈ [M]2. f x = 0 if M ⊆ WW card M = m for M
proof -
  have finite M
    using False m that by auto
  with not1 [simplified, rule-format, of M] that
  have ∃ x ∈ [M]2. f x ≠ Suc 0
    by (simp add: image-subset-iff finite-ordertype-eq-card m)
  with that show ?thesis

```

by (metis PiE f lessThan-iff less-2-cases nsets-mono numeral-2-eq-2  
 subset-iff)  
 qed  
 have m-fh-0:  $\exists x \in [M]^2. ?fh\ x = 0$  if  $M \subseteq WW$  card  $M = m$  for  $M$   
 proof –  
 have h '  $M \subseteq WW$   
 using  $\langle WW' \subseteq WW \rangle \langle \text{bij-betw } h\ WW\ WW' \rangle \text{bij-betwE that(1)}$  by fastforce  
 moreover have card (h '  $M$ ) = m  
 by (metis  $\langle \text{bij-betw } h\ WW\ WW' \rangle \text{bij-betw-def bij-betw-subset card-image}$   
 that)  
 ultimately have  $\exists x \in [h\ ' M]^2. f\ x = 0$   
 by (metis m-f-0)  
 then obtain Y where  $Y: f\ (h\ ' Y) = 0$   $Y \subseteq M$  and finite (h '  $Y$ ) card  
 (h '  $Y$ ) = 2  
 by (auto simp: nsets-def subset-image-iff)  
 then have card  $Y = 2$   
 using  $\langle \text{bij-betw } h\ WW\ WW' \rangle \langle M \subseteq WW \rangle$   
 by (metis bij-betw-def card-image inj-on-subset)  
 with Y card.infinite[of Y] show ?thesis  
 by (auto simp: nsets-def)  
 qed  
  
 obtain N j where infinite N  
 and N:  $\bigwedge k\ u. [k > 0; u \in [WW]^2; \text{Form } k\ u; [\text{enum } N\ k] < \text{inter-scheme}$   
 $k\ u; \text{List.set } (\text{inter-scheme } k\ u) \subseteq N] \implies ?fh\ u = j\ k$   
 using lemma-3-6 [OF fh] by blast  
  
 have infN': infinite (enum N ' {k<..}) for k  
 by (simp add: infinite N) enum-works finite-image-iff infinite-Ioi strict-mono-imp-inj-on)  
 have j-0:  $j\ k = 0$  if  $k > 0$  for k  
 proof –  
 obtain M where  $M: M \in [WW]^m$   
 and MF:  $\bigwedge u. u \in [M]^2 \implies \text{Form } k\ u$   
 and Mi:  $\bigwedge u. u \in [M]^2 \implies \text{List.set } (\text{inter-scheme } k\ u) \subseteq \text{enum } N\ '$   
 {k<..}  
 using lemma-3-7 [OF infN'  $\langle k > 0 \rangle$ ] by metis  
 obtain u where  $u: u \in [M]^2$  ?fh u = 0  
 using m-fh-0 [of M] M [unfolded nsets-def] by force  
 moreover  
 have §:  $\text{Form } k\ u$  List.set (inter-scheme k u)  $\subseteq \text{enum } N\ ' \{k<..\}$   
 by (simp-all add: MF Mi  $\langle u \in [M]^2 \rangle$ )  
 then have hd (inter-scheme k u)  $\in \text{enum } N\ ' \{k<..\}$   
 using hd-in-set inter-scheme-simple that by blast  
 then have [enum N k] < inter-scheme k u  
 using strict-mono-enum [OF  $\langle \text{infinite } N \rangle$ ] by (auto simp: less-list-def  
 strict-mono-def)  
 moreover have  $u \in [WW]^2$   
 using M u by (auto simp: nsets-def)  
 moreover have enum N ' {k<..}  $\subseteq N$

```

    using ⟨infinite N⟩ range-enum by auto
  ultimately show ?thesis
    using N § that by auto
qed
obtain X where X ⊆ WW and otX: ordertype X (lenlex less-than) = ω↑ω
  and X: ⋀u. u ∈ [X]2 ⇒
    ∃l. Form l u ∧ (l > 0 → [enum N l] < inter-scheme l u ∧ List.set
(inter-scheme l u) ⊆ N)
  using lemma-3-8 [OF ⟨infinite N⟩] ot' by blast
  have 0: ?fh ' [X]2 ⊆ {0}
  proof clarsimp
    fix u
    assume u: u ∈ [X]2
    obtain l where Form l u and l: l > 0 → [enum N l] < inter-scheme l u
  ∧ List.set (inter-scheme l u) ⊆ N
    using u X by blast
    have ?fh u = 0
    proof (cases l = 0)
      case True
      then show ?thesis
        by (metis Form-0-cases-raw ⟨Form l u⟩ ⟨X ⊆ WW⟩ doubleton-in-nsets-2
fh-eq-0-eqlen subset-iff u)
    next
      case False
      then obtain [enum N l] < inter-scheme l u List.set (inter-scheme l u) ⊆
N j l = 0
        using Nat.neq0-conv j-0 l by blast
      with False show ?thesis
        using ⟨X ⊆ WW⟩ N inter-scheme ⟨Form l u⟩ doubleton-in-nsets-2 u by
(auto simp: nsets-def)
    qed
    then show f (h ' u) = 0
      by auto
  qed
show ?P0
proof (intro exI conjI)
  show h ' X ⊆ WW
  using ⟨WW' ⊆ WW⟩ ⟨X ⊆ WW⟩ ⟨bij-betw h WW WW'⟩ bij-betw-imp-surj-on
by fastforce
  show ordertype (h ' X) (lenlex less-than) = ω ↑ ω
  proof (subst ordertype-inc-eq)
    show (h x, h y) ∈ lenlex less-than
      if x ∈ X y ∈ X (x, y) ∈ lenlex less-than for x y
      using that h ⟨X ⊆ WW⟩ by (auto simp: FR-WW FR-WW' iso-iff2
iso-ll-def)
  qed (use otX in auto)
  show f ' [h ' X]2 ⊆ {0}
  proof (clarsimp simp: image-subset-iff nsets-def)
    fix Y

```

```

assume  $Y: Y \subseteq h \text{ ' } X$  finite  $Y$   $\text{card } Y = 2$ 
then have inv-into  $WW$   $h \text{ ' } Y \subseteq X$ 
  by (metis  $\langle X \subseteq WW \rangle$   $\langle \text{bij-betw } h \text{ } WW \text{ } WW \rangle$  bij-betw-inv-into-LEFT
image-mono)
  moreover have finite (inv-into  $WW$   $h \text{ ' } Y$ )
    using  $\langle \text{finite } Y \rangle$  by blast
  moreover have  $\text{card} (\text{inv-into } WW \text{ } h \text{ ' } Y) = 2$ 
  using  $Y$  by (metis  $\langle X \subseteq WW \rangle$  card-image inj-on-inv-into subset-image-iff
subset-trans)
  ultimately have  $f (h \text{ ' } \text{inv-into } WW \text{ } h \text{ ' } Y) = 0$ 
    using  $0$  by (auto simp: image-subset-iff nsets-def)
  then show  $f Y = 0$ 
    by (metis  $\langle X \subseteq WW \rangle$   $\langle Y \subseteq h \text{ ' } X \rangle$  image-inv-into-cancel image-mono
order-trans)
  qed
qed
qed
then show  $\exists i < \text{Suc } 0. \exists H \subseteq WW. \text{ordertype } H \text{ } ?R = [\omega \uparrow \omega, \alpha] ! i \wedge f \text{ ' } [H]^2 \subseteq \{i\}$ 
  by (metis One-nat-def lessI nth-Cons-0 nth-Cons-Suc zero-less-Suc)
qed
qed

```

Theorem 3.1 of Jean A. Larson, *ibid*.

```

theorem partition- $\omega\omega$ :  $\alpha \in \text{elts } \omega \implies \text{partn-lst-VWF } (\omega \uparrow \omega) [\omega \uparrow \omega, \alpha] 2$ 
  using partn-lst-imp-partn-lst-VWF-eq [OF partition- $\omega\omega$ -aux] ordertype-WW by
auto
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. Erdős and E. C. Milner. A theorem in the partition calculus. *Canadian Mathematical Bulletin*, 15(4):501–505, Dec. 1972.
- [2] P. Erdős 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  $\omega^\omega$ . *Annals of Mathematical Logic*, 6(2):129–145, Dec. 1973.