The Nash-Williams Theorem

Lawrence C. Paulson

February 23, 2021

Abstract

In 1965, Nash-Williams [1] discovered a generalisation of the infinite form of Ramsey's theorem. Where the latter concerns infinite sets of n-element sets for some fixed n, the Nash-Williams theorem concerns infinite sets of finite sets (or lists) subject to a "no initial segment" condition. The present formalisation follows Todorčević [2].

Contents

1 The Pointwise Less-Than Relation Between Two Sets

theory *Nash-Extras*
imports *HOL−Library.Ramsey HOL−Library.Countable-Set*

begin

definition *less-sets* :: ['*a*::*order set*, '*a*::*order set*] ⇒ *bool* **where**
less-sets A B ≡ ∀ *x*∈*A*. ∀ *y*∈*B*. *x* < *y*

lemma *less-setsD*: ⟦*less-sets A B*; *a* ∈ *A*; *b* ∈ *B*⟧ ⟹ *a* < *b*
by (*auto simp*: *less-sets-def*)

lemma *less-sets-irrefl* [*simp*]: *less-sets A A* ⟷ *A* = {}
by (*auto simp*: *less-sets-def*)

lemma *less-sets-trans*: $[\![$*less-sets A B*; *less-sets B C*; $B \neq \{\}]\!] \Longrightarrow$ *less-sets A C*
unfolding *less-sets-def* **using** *less-trans* **by** *blast*

lemma *less-sets-weaken1*: $[\![$*less-sets A' B*; $A \subseteq A']\!] \Longrightarrow$ *less-sets A B*
by (*auto simp*: *less-sets-def*)

lemma *less-sets-weaken2*: $[\![$*less-sets A B'*; $B \subseteq B']\!] \Longrightarrow$ *less-sets A B*
by (*auto simp*: *less-sets-def*)

lemma *less-sets-imp-disjnt*: *less-sets A B* $\Longrightarrow$ *disjnt A B*
by (*auto simp*: *less-sets-def disjnt-def*)

lemma *less-sets-UN1*: *less-sets* $(\bigcup \mathcal{A})$ *B* $\longleftrightarrow$ $(\forall A{\in}\mathcal{A}.$ *less-sets A B*$)$
by (*auto simp*: *less-sets-def*)

lemma *less-sets-UN2*: *less-sets A* $(\bigcup \mathcal{B})$ $\longleftrightarrow$ $(\forall B{\in}\mathcal{B}.$ *less-sets A B*$)$
by (*auto simp*: *less-sets-def*)

lemma *less-sets-Un1*: *less-sets* $(A \cup A')$ *B* $\longleftrightarrow$ *less-sets A B* $\wedge$ *less-sets A' B*
by (*auto simp*: *less-sets-def*)

lemma *less-sets-Un2*: *less-sets A* $(B \cup B')$ $\longleftrightarrow$ *less-sets A B* $\wedge$ *less-sets A B'*
by (*auto simp*: *less-sets-def*)

lemma *strict-sorted-imp-less-sets*:
strict-sorted (*as* @ *bs*) $\Longrightarrow$ *less-sets* (*list.set as*) (*list.set bs*)
by (*simp add*: *less-sets-def sorted-wrt-append strict-sorted-sorted-wrt*)

lemma *Sup-nat-less-sets-singleton*:
fixes *n*::*nat*
assumes *Sup T* $<$ *n finite T*
shows *less-sets T* $\{n\}$
using *assms Max-less-iff*
by (*auto simp*: *Sup-nat-def less-sets-def split*: *if-split-asm*)

end

2 The Nash-Williams Theorem

Following S. Todorčević, *Introduction to Ramsey Spaces*, Princeton University Press (2010), 11–12.

theory *Nash-Williams*
imports *Nash-Extras*
begin

lemma *finite-nat-Int-greaterThan-iff*:
fixes *N* :: *nat set*
shows *finite* $(N \cap \{n{<}..\})$ $\longleftrightarrow$ *finite N*

apply (*simp add*: *finite-nat-iff-bounded subset-iff*)
by (*metis dual-order.strict-trans2 nat-less-le not-less-eq*)

2.1 Initial segments

definition *init-segment* :: *nat set* ⇒ *nat set* ⇒ *bool*
where *init-segment S T* ≡ ∃ *S'*. *T* = *S* ∪ *S'* ∧ *less-sets S S'*

lemma *init-segment-subset*: *init-segment S T* ⟹ *S* ⊆ *T*
by (*auto simp*: *init-segment-def*)

lemma *init-segment-refl*: *init-segment S S*
by (*metis empty-iff init-segment-def less-sets-def sup-bot.right-neutral*)

lemma *init-segment-antisym*: ⟦*init-segment S T*; *init-segment T S*⟧ ⟹ *S*=*T*
by (*auto simp*: *init-segment-def*)

lemma *init-segment-trans*: ⟦*init-segment S T*; *init-segment T U*⟧ ⟹ *init-segment S U*
unfolding *init-segment-def*
by (*meson UnE Un-assoc Un-upper1 less-sets-def less-sets-weaken1*)

lemma *init-segment-empty2* [*iff*]: *init-segment S* {} ⟷ *S*={}
by (*auto simp*: *init-segment-def less-sets-def*)

lemma *init-segment-Un*: *less-sets S S'* ⟹ *init-segment S* (*S* ∪ *S'*)
by (*auto simp*: *init-segment-def less-sets-def*)

lemma *init-segment-iff*:
shows *init-segment S T* ⟷ *S*=*T* ∨ (∃ *m* ∈ *T*. *S* = {*n* ∈ *T*. *n* < *m*}) (**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then obtain *S'* **where** *S'*: *T* = *S* ∪ *S' less-sets S S'*
by (*meson init-segment-def*)
then have *S* ⊆ *T*
by *auto*
then have *eq*: *S'* = *T*−*S*
using *S'* **by** (*auto simp*: *less-sets-def*)
show *?rhs*
proof (*cases T* ⊆ *S*)
case *True*
with ‹*S* ⊆ *T*› **show** *?rhs* **by** *blast*
next
case *False*
then have *Inf S'* ∈ *T*
by (*metis Diff-eq-empty-iff Diff-iff Inf-nat-def1 eq*)
moreover have ⋀*x*. *x* ∈ *S* ⟹ *x* < *Inf S'*
using *S' False* **by** (*auto simp*: *less-sets-def intro*!: *Inf-nat-def1*)

```
    moreover have {n ∈ T. n < Inf S'} ⊆ S
      using Inf-nat-def eq not-less-Least by fastforce
    ultimately show ?rhs
      using ‹S ⊆ T› by blast
  qed
next
  assume ?rhs
  then show ?lhs
  proof (elim disjE bexE)
    assume S = T
    then show init-segment S T
      using init-segment-refl by blast
  next
    fix m
    assume m: m ∈ T S = {n ∈ T. n < m}
    then have T = S ∪ {n ∈ T. m ≤ n}
      by auto
    moreover have less-sets S {n ∈ T. m ≤ n}
      using m by (auto simp: less-sets-def)
    ultimately show init-segment S T
      using init-segment-Un by force
  qed
qed

lemma init-segment-empty [iff]: init-segment {} S
  by (auto simp: init-segment-def less-sets-def)


lemma init-segment-insert-iff:
  assumes Sn: less-sets S {n} and TS: ⋀x. x ∈ T−S ⟹ n≤x
  shows init-segment (insert n S) T ⟷ init-segment S T ∧ n ∈ T
proof safe
  assume L: init-segment (insert n S) T
  then have init-segment ({n} ∪ S) T
    by auto
  then show init-segment S T
    by (metis (no-types) Sn init-segment-Un init-segment-trans sup.commute)
  show n ∈ T
    using L by (auto simp: init-segment-def)
next
  assume init-segment S T n ∈ T
  then obtain S' where S': T = S ∪ S' less-sets S S'
    by (auto simp: init-segment-def less-sets-def)
  then have S ∪ S' = insert n (S ∪ (S' − {n})) ∧
              less-sets (insert n S) (S' − {n})
    unfolding less-sets-def using ‹n ∈ T› TS nat-less-le by auto
  then show init-segment (insert n S) T
    using S'(1) init-segment-Un by force
qed
```

lemma *init-segment-insert*:
  assumes *init-segment S T* **and** *T*: *less-sets T* $\{n\}$
  shows *init-segment S* (*insert n T*)
proof (*cases T*=$\{\}$)
  case *True*
  then show *?thesis*
    using *assms*(*1*) **by** *blast*
next
  case *False*
  obtain S' **where** S': $T = S \cup S'$ *less-sets S S'*
    by (*meson assms init-segment-def*)
  then have *insert n T* = $S \cup$ (*insert n S'*) *less-sets S* (*insert n S'*)
    using *T False* **by** (*auto simp*: *less-sets-def*)
  then show *?thesis*
    using *init-segment-Un* **by** *presburger*
qed

2.2 Definitions and basic properties

definition *Ramsey* :: [*nat set set*, *nat*] ⇒ *bool*
  where *Ramsey* $\mathcal{F}$ *r* ≡ ∀*f* ∈ $\mathcal{F}$ → {..<*r*}.
                    ∀ *M*. *infinite M* ⟶
                        (∃ *N i*. *N* ⊆ *M* ∧ *infinite N* ∧ *i*<*r* ∧ (∀ *j*<*r*. *j*≠*i* ⟶ *f* −‘ {*j*} ∩ $\mathcal{F}$ ∩ *Pow N* = {}))

definition *thin-set* :: *nat set set* ⇒ *bool*
  where *thin-set* $\mathcal{F}$ ≡ $\mathcal{F}$ ⊆ *Collect finite* ∧ (∀ *S*∈$\mathcal{F}$. ∀ *T*∈$\mathcal{F}$. *init-segment S T* ⟶ *S*=*T*)

definition *comparables* :: *nat set* ⇒ *nat set* ⇒ *nat set set*
  where *comparables S M* ≡ {*T*. *finite T* ∧ (*init-segment T S* ∨ *init-segment S T* ∧ *T*−*S* ⊆ *M*)}

lemma *comparables-iff*: *T* ∈ *comparables S M* ⟷ *finite T* ∧ (*init-segment T S* ∨ *init-segment S T* ∧ *T* ⊆ *S* ∪ *M*)
  by (*auto simp*: *comparables-def init-segment-def*)

lemma *comparables-subset*: ⋃(*comparables S M*) ⊆ *S* ∪ *M*
  by (*auto simp*: *comparables-def init-segment-def*)

lemma *comparables-empty* [*simp*]: *comparables* {} *M* = *Fpow M*
  by (*auto simp*: *comparables-def Fpow-def*)

lemma *comparables-mono*: *N* ⊆ *M* ⟹ *comparables S N* ⊆ *comparables S M*
  by (*auto simp*: *comparables-def*)

definition *rejects* $\mathcal{F}$ *S M* ≡ *comparables S M* ∩ $\mathcal{F}$ = {}

abbreviation *accepts*
  where *accepts* $\mathcal{F}$ *S M* $\equiv \neg$ *rejects* $\mathcal{F}$ *S M*

definition *strongly-accepts*
  where *strongly-accepts* $\mathcal{F}$ *S M* $\equiv (\forall N{\subseteq}M.$ *rejects* $\mathcal{F}$ *S N* $\longrightarrow$ *finite N*)

definition *decides*
  where *decides* $\mathcal{F}$ *S M* $\equiv$ *rejects* $\mathcal{F}$ *S M* $\vee$ *strongly-accepts* $\mathcal{F}$ *S M*

definition *decides-subsets*
  where *decides-subsets* $\mathcal{F}$ *M* $\equiv \forall T.\ T \subseteq M \longrightarrow$ *finite T* $\longrightarrow$ *decides* $\mathcal{F}$ *T M*

lemma *strongly-accepts-imp-accepts*:
  $[\![$*strongly-accepts* $\mathcal{F}$ *S M*; *infinite M*$]\!] \Longrightarrow$ *accepts* $\mathcal{F}$ *S M*
  unfolding *strongly-accepts-def* **by** *blast*

lemma *rejects-trivial*: $[\![$*rejects* $\mathcal{F}$ *S M*; *thin-set* $\mathcal{F}$; *init-segment F S*; $F \in \mathcal{F}]\!] \Longrightarrow$ *False*
  unfolding *rejects-def thin-set-def*
  using *comparables-iff* **by** *blast*

lemma *rejects-subset*: $[\![$*rejects* $\mathcal{F}$ *S M*; $N \subseteq M]\!] \Longrightarrow$ *rejects* $\mathcal{F}$ *S N*
  by (*fastforce simp add*: *rejects-def comparables-def*)

lemma *strongly-accepts-subset*: $[\![$*strongly-accepts* $\mathcal{F}$ *S M*; $N \subseteq M]\!] \Longrightarrow$ *strongly-accepts* $\mathcal{F}$ *S N*
  by (*auto simp*: *strongly-accepts-def*)

lemma *decides-subset*: $[\![$*decides* $\mathcal{F}$ *S M*; $N \subseteq M]\!] \Longrightarrow$ *decides* $\mathcal{F}$ *S N*
  unfolding *decides-def*
  using *rejects-subset strongly-accepts-subset* **by** *blast*

lemma *decides-subsets-subset*: $[\![$*decides-subsets* $\mathcal{F}$ *M*; $N \subseteq M]\!] \Longrightarrow$ *decides-subsets* $\mathcal{F}$ *N*
  by (*meson decides-subset decides-subsets-def subset-trans*)

lemma *rejects-empty* [*simp*]: *rejects* $\mathcal{F}$ $\{\}$ *M* $\longleftrightarrow$ *Fpow M* $\cap$ $\mathcal{F} = \{\}$
  by (*auto simp*: *rejects-def comparables-def Fpow-def*)

lemma *strongly-accepts-empty* [*simp*]: *strongly-accepts* $\mathcal{F}$ $\{\}$ *M* $\longleftrightarrow$ $(\forall N{\subseteq}M.$ *Fpow N* $\cap$ $\mathcal{F} = \{\} \longrightarrow$ *finite N*)
  by (*simp add*: *strongly-accepts-def Fpow-def disjoint-iff*)

lemma *ex-infinite-decides-1*:
  assumes *infinite M*
  obtains *N* **where** $N \subseteq M$ *infinite N decides* $\mathcal{F}$ *S N*
  by (*meson assms decides-def order-refl strongly-accepts-def*)

proposition *ex-infinite-decides-finite*:

assumes *infinite M finite S*
obtains *N* **where** $N \subseteq M$ *infinite N* $\bigwedge T.\ T \subseteq S \Longrightarrow$ *decides* $\mathcal{F}$ *T N*
proof −
have *finite* (*Pow S*)
by (*simp add*: ‹*finite S*›)
then obtain *f* :: *nat* ⇒ *nat set* **where** *f*: *f* ‘ {..< *card* (*Pow S*)} = *Pow S*
by (*metis bij-betw-imp-surj-on* [*OF bij-betw-from-nat-into-finite*])
obtain *M0* **where** *M0*: *infinite M0 M0* ⊆ *M decides* $\mathcal{F}$ (*f 0*) *M0*
by (*meson* ‹*infinite M*› *ex-infinite-decides-1*)
define *F* **where** *F* ≡ *rec-nat M0* (λ*n N*. @*N′*. *N′* ⊆ *N* ∧ *infinite N′* ∧ *decides* $\mathcal{F}$ (*f* (*Suc n*)) *N′*)
have *P-Suc*: *F* (*Suc n*) = (@*N′*. *N′* ⊆ *F n* ∧ *infinite N′* ∧ *decides* $\mathcal{F}$ (*f* (*Suc n*)) *N′*) **for** *n*
by (*auto simp*: *F-def*)
have ∗: *infinite* (*F n*) ∧ *decides* $\mathcal{F}$ (*f n*) (*F n*) ∧ *F n* ⊆ *M* **for** *n*
proof (*induction n*)
case *0*
with ‹*infinite M*› **show** *?case*
by (*auto simp*: *F-def M0*)
next
case (*Suc n*)
let *?*Φ = λ*N′*. *N′* ⊆ *F n* ∧ *infinite N′* ∧ *decides* $\mathcal{F}$ (*f* (*Suc n*)) *N′*
have ∃ *N′*. *?*Φ *N′*
by (*meson Suc ex-infinite-decides-1 subset-trans*)
then have *Eps ?*Φ ⊆ *F n* ∧ *infinite* (*Eps ?*Φ) ∧ *decides* $\mathcal{F}$ (*f* (*Suc n*)) (*Eps ?*Φ)
by (*rule someI-ex*)
with *Suc.IH* **show** *?case*
by (*auto simp*: *P-Suc*)
qed
then have *telescope*: *F* (*Suc n*) ⊆ *F n* **for** *n*
unfolding *P-Suc* **by** (*metis* (*no-types*, *lifting*) *ex-infinite-decides-1 someI-ex*)
let *?N* = $\bigcap$ *n*<*card* (*Pow S*). *F n*
show *thesis*
proof
show *?N* ⊆ *M*
by (*metis* ∗ *INF-lower2 Pow-iff f imageE order-refl*)
next
have *eq*: ($\bigcap$ *n*≤*m*. *F n*) = *F m* **for** *m*
by (*induction m*) (*use telescope* **in** ‹*auto simp*: *atMost-Suc*›)
then show *infinite ?N*
by (*metis* ∗ *Suc-le-D Suc-le-eq finite-subset le-INF-iff lessThan-Suc-atMost lessThan-iff*)
next
fix *T*
assume *T* ⊆ *S*
then obtain *m* **where** *f m* = *T m* < *card* (*Pow S*)
using *f* **by** (*blast elim*: *equalityE*)
then have *decides* $\mathcal{F}$ *T* (*F m*)

using $*$ **by** *blast*
show *decides* $\mathcal{F}$ *T ?N*
by (*meson INT-lower* ‹*decides* $\mathcal{F}$ *T* (*F m*)› ‹*m* < *card* (*Pow S*)› *decides-subset lessThan-iff*)
qed
qed

lemma *sorted-wrt-subset*: ⟦*X* ∈ *list.set l*; *sorted-wrt* (≤) *l*⟧ ⟹ *hd l* ⊆ *X*
by (*induction l*) *auto*

Todorčević's Lemma 1.18

proposition *ex-infinite-decides-subsets*:
assumes *thin-set* $\mathcal{F}$ *infinite M*
obtains *N* **where** *N* ⊆ *M infinite N decides-subsets* $\mathcal{F}$ *N*
proof −
obtain *M0* **where** *M0*: *infinite M0 M0* ⊆ *M decides* $\mathcal{F}$ {} *M0*
by (*meson* ‹*infinite M*› *ex-infinite-decides-1*)
define *decides-all* **where** *decides-all* ≡ λ*S N*. ∀ *T* ⊆ *S*. *decides* $\mathcal{F}$ *T N*
define Φ **where** Φ ≡ λ*NL N*. *N* ⊆ *hd NL* ∧ *Inf N* > *Inf* (*hd NL*) ∧ *infinite N* ∧ *decides-all* (*List.set* (*map Inf NL*)) *N*
have ∃ *N*. Φ *NL N* **if** *infinite* (*hd NL*) **for** *NL*
proof −
obtain *N* **where** *N*: *N* ⊆ *hd NL* ∧ *infinite N* ∧ *decides-all* (*List.set* (*map Inf NL*)) *N*
unfolding Φ*-def decides-all-def*
by (*metis List.finite-set ex-infinite-decides-finite* ‹*infinite* (*hd NL*)›)
then have *Inf* (*N* ∩ {*Inf* (*hd NL*)<..}) > *Inf* (*hd NL*)
by (*metis Inf-nat-def1 Int-iff finite.emptyI finite-nat-Int-greaterThan-iff greaterThan-iff*)
then show *?thesis*
unfolding Φ*-def decides-all-def*
by (*meson Int-lower1 N decides-all-def decides-subset finite-nat-Int-greaterThan-iff subset-trans*)
qed
then have Φ*-Eps*: Φ *NL* (*Eps* (Φ *NL*)) **if** *infinite* (*hd NL*) **for** *NL*
by (*simp add*: *someI-ex that*)
define *F* **where** *F* ≡ *rec-nat* [*M0*] (λ*n NL*. (*Eps* (Φ *NL*)) # *NL*)
have *F-simps* [*simp*]: *F 0* = [*M0*] *F* (*Suc n*) = *Eps* (Φ (*F n*)) # *F n* **for** *n*
by (*auto simp*: *F-def*)
have *F*: *F n* ≠ [] ∧ *sorted-wrt* (≤) (*F n*) ∧ *list.set* (*F n*) ⊆ *Collect infinite* ∧ *list.set* (*F n*) ⊆ *Pow M* **for** *n*
proof (*induction n*)
case *0*
then show *?case*
by (*simp add*: *M0*)
next
case (*Suc n*)
then have $*$: Φ (*F n*) (*Eps* (Φ (*F n*)))

```
    using Φ-Eps hd-in-set by blast
  show ?case
  proof (intro conjI)
    show sorted-wrt (⊆) (F (Suc n))
      using subset-trans [OF - sorted-wrt-subset] Suc.IH Φ-def ∗ by auto
    show list.set (F (Suc n)) ⊆ {S. infinite S}
      using ∗ Φ-def Suc.IH by force
    show list.set (F (Suc n)) ⊆ Pow M
      using ∗ Suc.IH Φ-def hd-in-set by force
  qed auto
 qed
 have ΦF: Φ (F n) (Eps (Φ (F n))) for n
  using F Φ-Eps hd-in-set by blast
 then have decides: decides-all (List.set (map Inf (F n))) (Eps (Φ (F n))) for n
  using Φ-def by blast
 have Eps-subset-hd: Eps (Φ (F n)) ⊆ hd (F n)  for n
  using ΦF Φ-def by blast
 have List.set (map Inf (F n)) ⊆ List.set (map Inf (F (Suc n))) for n
  by auto
 then have map-Inf-subset: m ≤ n ⟹ List.set (map Inf (F m)) ⊆ List.set (map
Inf (F n)) for m n
  by (rule order-class.lift-Suc-mono-le) auto
 define mmap where mmap ≡ λn. Inf (hd (F (Suc n)))
 have mmap n < mmap (Suc n) for n
  by (metis F-simps(2) ΦF Φ-def list.sel(1) mmap-def)
 then have strict-mono mmap
  by (simp add: lift-Suc-mono-less strict-mono-def)
 then have inj mmap
  by (simp add: strict-mono-imp-inj-on)
 have finite-F-bound: ∃ n. S ⊆ List.set (map Inf (F n))
  if S: S ⊆ range mmap finite S for S
 proof −
  obtain K where finite K S ⊆ mmap ‘ K
   by (metis S finite-subset-image order-refl)
  show ?thesis
  proof
   have mmap ‘ K ⊆ list.set (map Inf (F (Suc (Max K))))
    unfolding mmap-def image-subset-iff
    by (metis F Max-ge ‹finite K› hd-in-set imageI map-Inf-subset not-less-eq-eq
set-map subsetD)
   with S show S ⊆ list.set (map Inf (F (Suc (Max K))))
    using ‹S ⊆ mmap ‘ K› by auto
  qed
 qed
 have Eps (Φ (F (Suc n))) ⊆ Eps (Φ (F n)) for n
  by (metis F-simps(2) ΦF Φ-def list.sel(1))
 then have Eps-Φ-decreasing: m ≤ n ⟹ Eps (Φ (F n)) ⊆ Eps (Φ (F m)) for m
n
  by (rule order-class.lift-Suc-antimono-le)
```

```
  have hd-Suc-eq-Eps: hd (F (Suc n)) = Eps (Φ (F n)) for n
    by simp
  have Inf-hd-in-hd: Inf (hd (F n)) ∈ hd (F n) for n
    by (metis Inf-nat-def1 ΦF Φ-def finite.emptyI rev-finite-subset)
  then have Inf-hd-in-Eps: Inf (hd (F m)) ∈ Eps (Φ (F n)) if m>n for m n
    by (metis Eps-Φ-decreasing Nat.lessE leD mmap-def not-less-eq-eq subsetD that
hd-Suc-eq-Eps)
  then have image-mmap-subset-hd-F: mmap ‘ {n..} ⊆ hd (F (Suc n)) for n
    by (metis hd-Suc-eq-Eps atLeast-iff image-subsetI le-imp-less-Suc mmap-def)
  have list.set (F k) ⊆ list.set (F n) if k ≤ n for k n
    by (rule order-class.lift-Suc-mono-le) (use that in auto)
  then have hd-F-in-F: hd (F k) ∈ list.set (F n) if k ≤ n for k n
    by (meson F hd-in-set subsetD that)
  show thesis
  proof
    show infinite-mm: infinite (range mmap)
      using ‹inj mmap› range-inj-infinite by blast
    show range mmap ⊆ M
      using Eps-subset-hd ‹M0 ⊆ M› image-mmap-subset-hd-F by fastforce
    show decides-subsets 𝓕 (range mmap)
      unfolding decides-subsets-def
    proof (intro strip)
      fix S
      assume S ⊆ range mmap finite S
      define n where n ≡ LEAST n. S ⊆ List.set (map Inf (F n))
      have ∃ n. S ⊆ List.set (map Inf (F n))
        using ‹S ⊆ range mmap› ‹finite S› finite-F-bound by blast
      then have S: S ⊆ List.set (map Inf (F n)) and minS: ⋀m. m<n ⟹ ¬ S
⊆ List.set (map Inf (F m))
        unfolding n-def by (meson LeastI-ex not-less-Least)+
      have decides-Fn: decides 𝓕 S (Eps (Φ (F n)))
        using S decides decides-all-def by blast
      show decides 𝓕 S (range mmap)
      proof (cases n=0)
        case True
        then show ?thesis
        by (metis image-mmap-subset-hd-F decides-Fn decides-subset hd-Suc-eq-Eps
atLeast-0)
      next
        case False
        have notin-map-Inf: x ∉ List.set (map Inf (F n)) if less-sets S {x} for x
        proof clarsimp
          fix T
          assume x = Inf T and T ∈ list.set (F n)
          with that have ls: less-sets S {Inf T}
            by auto
          have S ⊆ List.set (map Inf (F j))
            if  T: T ∈ list.set (F (Suc j)) for j
          proof clarsimp
```

```
      fix x
      assume x ∈ S
      then have x < Inf T
        using less-setsD ls by blast
      then have x ∉ T
        using Inf-nat-def not-less-Least by auto
      obtain k where k: x = mmap k
        using ‹S ⊆ range mmap› ‹x ∈ S› by blast
      with T ‹x < Inf T› have k < j
          by (metis Eps-Φ-decreasing F Inf-hd-in-hd hd-Suc-eq-Eps ‹x ∉ T›
mmap-def not-le sorted-wrt-subset subsetD)
      then have Eps (Φ (F k)) ∈ list.set (F j)
        by (metis Suc-leI hd-Suc-eq-Eps hd-F-in-F)
      then show x ∈ Inf ‘ list.set (F j)
        by (auto simp: k image-iff mmap-def)
    qed
    then obtain m where m<n S ⊆ List.set (map Inf (F m))
      using ‹n ≠ 0› by (metis ‹T ∈ list.set (F n)› lessI less-Suc-eq-0-disj)
    then show False
      using minS by blast
  qed
  have Inf-hd-F: Inf (hd (F m)) ∈ Eps (Φ (F n)) if less-sets S {Inf (hd (F
m))} for m
    by (metis Inf-hd-in-Eps hd-F-in-F notin-map-Inf imageI leI set-map that)
  have less-S: less-sets S {Inf (hd (F m))}
    if init-segment S T Inf (hd (F m)) ∈ T Inf (hd (F m)) ∉ S for T m
    using ‹finite S› that by (auto simp: init-segment-iff less-sets-def)
   consider rejects F S (Eps (Φ (F n))) | strongly-accepts F S (Eps (Φ (F
n)))
    using decides-Fn by (auto simp: decides-def)
  then show ?thesis
  proof cases
    case 1
    have rejects F S (range mmap)
    proof (clarsimp simp: rejects-def disjoint-iff)
      fix X
      assume X ∈ comparables S (range mmap) and X ∈ F
       moreover have ⋀x X. ⟦X ∈ F; init-segment S X; x ∈ X; x ∉ S; x ∈
range mmap⟧
                          ⟹ x ∈ Eps (Φ (F n))
        using less-S Inf-hd-F mmap-def by blast
      ultimately have X ∈ comparables S (Eps (Φ (F n)))
        by (auto simp: comparables-def disjoint-iff subset-iff)
      with 1 ‹X ∈ F› show False by (auto simp: rejects-def)
    qed
    then show ?thesis
      by (auto simp: decides-def)
  next
    case 2
```

```
      have False
        if N ⊆ range mmap and rejects ℱ S N and infinite N for N
      proof −
        have N = mmap ‘ {n..} ∩ N ∪ mmap ‘ {..<n} ∩ N
          using in-mono that(1) by fastforce
        then have infinite (mmap ‘ {n..} ∩ N)
          by (metis finite-Int finite-Un finite-imageI finite-lessThan that(3))
        moreover have rejects ℱ S (mmap ‘ {n..} ∩ N)
          using rejects-subset ‹rejects ℱ S N› by fastforce
        moreover have mmap ‘ {n..} ∩ N ⊆Eps (Φ (F n))
          using image-mmap-subset-hd-F by fastforce
        ultimately show ?thesis
          using 2 by (auto simp: strongly-accepts-def)
      qed
      with 2 have strongly-accepts ℱ S (range mmap)
        by (auto simp: strongly-accepts-def)
      then show ?thesis
        by (auto simp: decides-def)
    qed
  qed
 qed
qed
```

Todorčević's Lemma 1.19

```
proposition strongly-accepts-1-19:
  assumes acc: strongly-accepts ℱ S M
    and thin-set ℱ infinite M S ⊆ M finite S
    and dsM: decides-subsets ℱ M
  shows finite {n ∈ M. ¬ strongly-accepts ℱ (insert n S) M}
proof (rule ccontr)
  define N where N = {n ∈ M. rejects ℱ (insert n S) M} ∩ {Sup S<..}
  have N ⊆ M and N: ⋀n. n ∈ N ⟷ n ∈ M ∧ rejects ℱ (insert n S) M ∧ n
> Sup S
    by (auto simp: N-def)
  assume ¬ ?thesis
   moreover have {n ∈ M. ¬ strongly-accepts ℱ (insert n S) M} = {n ∈ M.
rejects ℱ (insert n S) M}
    using dsM ‹finite S› ‹infinite M› ‹S ⊆ M› unfolding decides-subsets-def
    by (meson decides-def finite.insertI insert-subset strongly-accepts-imp-accepts)
  ultimately have infinite {n ∈ M. rejects ℱ (insert n S) M}
    by simp
  then have infinite N
    by (simp add: N-def finite-nat-Int-greaterThan-iff)
  then have accepts ℱ S N
    using acc strongly-accepts-def ‹N ⊆ M› by blast
  then obtain T where T: T ∈ comparables S N T ∈ ℱ and TSN: T ⊆ S ∪ N
    unfolding rejects-def
    using comparables-iff init-segment-subset by fastforce
```

then consider *init-segment T S* | *init-segment S T S≠T*
by (*auto simp*: *comparables-def*)
then show *False*
proof *cases*
case *1*
then have *init-segment T* (*insert n S*) **if** $n \in N$ **for** *n*
by (*meson Sup-nat-less-sets-singleton N* ‹*finite S*› *init-segment-insert that*)
with ‹*infinite N*› ‹*thin-set* $\mathcal{F}$› ‹$T \in \mathcal{F}$› **show** *False*
by (*meson N infinite-nat-iff-unbounded rejects-trivial*)
next
let *?n* = *Min* (*T*−*S*)
case *2*
then have *?n* ∈ *N*
by (*metis Diff-partition Diff-subset-conv Min-in T*(*1*) *TSN comparables-iff finite-Diff init-segment-subset subsetD sup-bot.right-neutral*)
then have *rejects* $\mathcal{F}$ (*insert ?n S*) *N*
using *rejects-subset* ‹*N* ⊆ *M*› **by** (*auto simp*: *N-def*)
then have §: ¬ *init-segment T* (*insert ?n S*) ∧ (¬ *init-segment* (*insert ?n S*) *T* ∨ *insert ?n S* = *T*)
using *T Diff-partition TSN* ‹*Min* (*T* − *S*) ∈ *N*› ‹*finite S*›
unfolding *rejects-def comparables-iff disjoint-iff*
by *auto*
then have *T-nS*: *T* ⊆ *insert ?n S*
proof (*elim conjE disjE*)
assume ¬ *init-segment T* (*insert ?n S*) ¬ *init-segment* (*insert ?n S*) *T*
moreover have *less-sets S* {*Min* (*T* − *S*)}
using *Sup-nat-less-sets-singleton N* ‹*Min* (*T* − *S*) ∈ *N*› *assms*(*5*) **by** *blast*
moreover have *finite* (*T* − *S*)
using *T comparables-iff* **by** *blast*
ultimately show *?thesis*
using ‹*init-segment S T*› *Min-in init-segment-insert-iff* **by** *auto*
qed *auto*
have *non-TS*: ¬ *init-segment T S*
by (*meson Sup-nat-less-sets-singleton N* ‹*?n* ∈ *N*› ‹¬ *init-segment T* (*insert* (*?n*) *S*) ∧ (¬ *init-segment* (*insert* (*?n*) *S*) *T* ∨ *insert* (*?n*) *S* = *T*)› *assms*(*5*) *init-segment-insert*)
consider (*ST*) *S* ⊆ *T* | (*TS*) *T* ⊆ *S*
using *2 init-segment-subset* **by** *blast*
then show *False*
proof *cases*
case *ST*
with § **show** *?thesis*
using *2 T*(*1*) ‹*T* ⊆ *insert* (*?n*) *S*› *comparables-iff init-segment-iff* **by** *auto*
next
case *TS*
then show *?thesis*
proof −
have ¬ *init-segment T S*
by (*meson Sup-nat-less-sets-singleton N* ‹*?n* ∈ *N*› § *assms*(*5*) *init-segment-insert*)

```
      then show ?thesis
        using 2 TS init-segment-subset by fastforce
    qed
  qed
 qed
qed
```

Much work is needed for this slight strengthening of the previous result!

```
proposition strongly-accepts-1-19-plus:
 assumes thin-set ℱ infinite M
   and dsM: decides-subsets ℱ M
 obtains N where N ⊆ M infinite N
     ⋀S n. ⟦S ⊆ N; finite S; strongly-accepts ℱ S N; n ∈ N; less-sets S {n}⟧
           ⟹ strongly-accepts ℱ (insert n S) N
proof −
 define insert-closed where
   insert-closed ≡ λNL N. ∀ T ⊆ Inf ‘ set NL. ∀ n ∈ N.
                  strongly-accepts ℱ T ((Inf ‘ set NL) ∪ hd NL) ⟶
                   less-sets T {n} ⟶ strongly-accepts ℱ (insert n T) ((Inf ‘ set
NL) ∪ hd NL)
 define Φ where Φ ≡ λNL N. N ⊆ hd NL ∧ Inf N > Inf (hd NL) ∧ infinite N
∧ insert-closed NL N
 have ∃ N. Φ NL N if NL: infinite (hd NL) Inf ‘ set NL ∪ hd NL ⊆ M for NL
 proof −
   let ?m = Inf ‘ set NL
   let ?M = ?m ∪ hd NL
   define P where P ≡ λS. {n ∈ ?M. ¬ strongly-accepts ℱ (insert n S) ?M}
   have ∃ k. P S ⊆ {..k}
     if S ⊆ Inf ‘ set NL strongly-accepts ℱ S ?M for S
   proof −
     have finite (P S)
       unfolding P-def
     proof (rule strongly-accepts-1-19)
       show decides-subsets ℱ ?M
         using NL(2) decides-subsets-subset dsM by blast
       show finite S
         using finite-surj that(1) by blast
     qed (use that NL assms in auto)
     then show ?thesis
       by (simp add: finite-nat-iff-bounded-le)
   qed
   then obtain f where f: ⋀S. ⟦S ⊆ Inf ‘ set NL; strongly-accepts ℱ S ?M⟧ ⟹
P S ⊆ {..f S}
     by metis
   define m where m ≡ Max (insert (Inf (hd NL)) (f ‘ Pow (Inf ‘ set NL)))
   have §: strongly-accepts ℱ (insert n S) ?M
     if S: S ⊆ Inf ‘ set NL strongly-accepts ℱ S ?M and n: n ∈ hd NL ∩ {m<..}
for S n
   proof −
```

```
    have f S ≤ m
      unfolding m-def using that(1) by auto
    then show ?thesis
      using f [OF S] n unfolding P-def by auto
  qed
  have Φ NL (hd NL ∩ {m<..})
    unfolding Φ-def
  proof (intro conjI)
  have Inf (hd NL) ≤ m
    by (simp add: m-def)
  then show Inf (hd NL) < Inf (hd NL ∩ {m<..})
      by (metis Inf-nat-def1 Int-iff finite.emptyI finite-nat-Int-greaterThan-iff
greaterThan-iff le-less-trans that(1))
    show infinite (hd NL ∩ {m<..})
      by (simp add: finite-nat-Int-greaterThan-iff that(1))
    show insert-closed NL (hd NL ∩ {m<..})
      by (auto intro: § simp: insert-closed-def)
  qed auto
  then show ?thesis ..
 qed
 then have Φ-Eps: Φ NL (Eps (Φ NL)) if infinite (hd NL)  Inf ‘ set NL ∪ hd NL
⊆ M for NL
  by (meson someI-ex that)
 define F where F ≡ rec-nat [M] (λn NL. (Eps (Φ NL)) # NL)
 have F-simps [simp]: F 0 = [M] F (Suc n) = Eps (Φ (F n)) # F n for n
  by (auto simp: F-def)
 have InfM: Inf M ∈ M
  by (metis Inf-nat-def1 assms(2) finite.emptyI)
 have F: F n ≠ [] ∧ sorted-wrt (≤) (F n) ∧ list.set (F n) ⊆ Collect infinite ∧ set
(F n) ⊆ Pow M ∧ Inf ‘ list.set (F n) ⊆ M for n
 proof (induction n)
  case 0
  then show ?case
   by (auto simp: InfM ‹infinite M›)
 next
  case (Suc n)
  have hd (F n) ⊆ M
   by (meson Pow-iff Suc.IH hd-in-set subsetD)
  then have 1: Ball (list.set (F n)) ((⊆) (Eps (Φ (F n))))
   using order-trans [OF - sorted-wrt-subset]
   by (metis Suc.IH Un-subset-iff Φ-Eps Φ-def hd-in-set mem-Collect-eq subsetD)
  have 2: infinite (Eps (Φ (F n)))
    by (metis Ball-Collect Pow-iff Suc.IH Un-subset-iff Φ-Eps Φ-def hd-in-set
subset-iff)
  have 3: Eps (Φ (F n)) ⊆ M
   by (meson Pow-iff Suc.IH 1 hd-in-set subset-iff)
  have Inf (Eps (Φ (F n))) ∈ M
   by (metis 2 3 Inf-nat-def1 finite.simps in-mono)
  with 1 2 3 show ?case
```

using *Suc* **by** *simp*
qed
have ΦF: $\Phi\ (F\ n)\ (Eps\ (\Phi\ (F\ n)))$ **for** n
by (*metis Ball-Collect F Pow-iff Un-subset-iff* Φ*-Eps hd-in-set subset-code*(*1*))
then have *insert-closed*: *insert-closed* $(F\ n)\ (Eps\ (\Phi\ (F\ n)))$ **for** n
using Φ*-def* **by** *blast*
have *Eps-subset-hd*: $Eps\ (\Phi\ (F\ n)) \subseteq hd\ (F\ n)$ **for** n
using ΦF Φ*-def* **by** *blast*
define *mmap* **where** $mmap \equiv \lambda n.\ Inf\ (hd\ (F\ (Suc\ n)))$
have $mmap\ n < mmap\ (Suc\ n)$ **for** n
by (*metis F-simps*(*2*) ΦF Φ*-def list.sel*(*1*) *mmap-def*)
then have *strict-mono mmap*
by (*simp add*: *lift-Suc-mono-less strict-mono-def*)
then have *inj mmap*
by (*simp add*: *strict-mono-imp-inj-on*)
have $Eps\ (\Phi\ (F\ (Suc\ n))) \subseteq Eps\ (\Phi\ (F\ n))$ **for** n
by (*metis F-simps*(*2*) ΦF Φ*-def list.sel*(*1*))
then have *Eps-*Φ*-decreasing*: $m \leq n \Longrightarrow Eps\ (\Phi\ (F\ n)) \subseteq Eps\ (\Phi\ (F\ m))$ **for** m n
by (*rule order-class.lift-Suc-antimono-le*)
have *hd-Suc-eq-Eps*: $hd\ (F\ (Suc\ n)) = Eps\ (\Phi\ (F\ n))$ **for** n
by *simp*
have $Inf\ (hd\ (F\ n)) \in hd\ (F\ n)$ **for** n
by (*metis Inf-nat-def1* ΦF Φ*-def finite.emptyI rev-finite-subset*)
then have *Inf-hd-in-Eps*: $Inf\ (hd\ (F\ m)) \in Eps\ (\Phi\ (F\ n))$ **if** $m{>}n$ **for** $m\ n$
by (*metis Eps-*Φ*-decreasing Nat.lessE hd-Suc-eq-Eps less-imp-le-nat subsetD that*)
then have *image-mmap-subset-hd-F*: $mmap\ `\ \{n..\} \subseteq hd\ (F\ (Suc\ n))$ **for** n
by (*metis hd-Suc-eq-Eps atLeast-iff image-subsetI le-imp-less-Suc mmap-def*)
have $list.set\ (F\ k) \subseteq list.set\ (F\ n)$ **if** $k \leq n$ **for** $k\ n$
by (*rule order-class.lift-Suc-mono-le*) (*use that* **in** *auto*)
then have *hd-F-in-F*: $hd\ (F\ k) \in list.set\ (F\ n)$ **if** $k \leq n$ **for** $k\ n$
by (*meson F hd-in-set subsetD that*)
show *?thesis*
proof
show *infinite-mm*: *infinite* (*range mmap*)
using ‹*inj mmap*› *range-inj-infinite* **by** *blast*
show $range\ mmap \subseteq M$
using *Eps-subset-hd image-mmap-subset-hd-F* **by** *fastforce*
next
fix $S\ a$
assume $S \subseteq range\ mmap$ *finite* S **and** *acc*: *strongly-accepts* $\mathcal{F}\ S$ (*range mmap*)
and a: $a \in range\ mmap$ **and** *Sn*: *less-sets* $S\ \{a\}$
then obtain n **where** n: $a = mmap\ n$
by *auto*
define N **where** $N \equiv LEAST\ n.\ S \subseteq mmap\ `\ \{..{<}n\}$
have $\exists n.\ S \subseteq mmap\ `\ \{..{<}n\}$
by (*metis* ‹$S \subseteq range\ mmap$› ‹*finite* S› *finite-nat-iff-bounded finite-subset-image image-mono*)

```
    then have S: S ⊆ mmap ` {..<N} and minS: ⋀m. m<N ⟹ ¬ S ⊆ mmap ` {..<m}
      unfolding N-def by (meson LeastI-ex not-less-Least)+
    have range-mmap-subset: range mmap ⊆ Inf ` list.set (F n) ∪ hd (F n) for n
    proof (induction n)
      case 0
      then show ?case
        using Eps-subset-hd image-mmap-subset-hd-F by fastforce
    next
      case (Suc n)
      then show ?case
        by clarsimp (metis Inf-hd-in-Eps hd-F-in-F image-iff leI mmap-def)
    qed
    have subM: (Inf ` list.set (F N) ∪ hd (F N)) ⊆ M
      by (meson F PowD hd-in-set subsetD sup.boundedI)
    have strongly-accepts F (insert a S) (Inf ` list.set (F N) ∪ hd (F N))
    proof (rule insert-closed [unfolded insert-closed-def, rule-format])
      have mmap ` {..<N} ⊆ Inf ` list.set (F N)
        using Suc-leI hd-F-in-F by (fastforce simp: mmap-def le-eq-less-or-eq)
      with S show Ssub: S ⊆ Inf ` list.set (F N)
        by auto
      have S ⊆ mmap ` {..<n}
        using Sn S ‹strict-mono mmap› strict-mono-less
        by (fastforce simp: less-sets-def n  image-iff subset-iff Bex-def)
      with leI minS have n≥N by blast
      then show a ∈ Eps (Φ (F N))
        using image-mmap-subset-hd-F n by fastforce
      show strongly-accepts F S (Inf ` list.set (F N) ∪ hd (F N))
      proof (rule ccontr)
        assume ¬ strongly-accepts F S (Inf ` list.set (F N) ∪ hd (F N))
        then have rejects F S (Inf ` list.set (F N) ∪ hd (F N))
          using dsM subM unfolding decides-subsets-def
          by (meson F Ssub ‹finite S› decides-def decides-subset subset-trans)
        moreover have accepts F S (range mmap)
          using ‹inj mmap› acc range-inj-infinite strongly-accepts-imp-accepts by
blast
        ultimately show False
          by (meson range-mmap-subset rejects-subset)
      qed
      show less-sets S {a}
        by (auto simp: Sn)
    qed
    then show strongly-accepts F (insert a S) (range mmap)
      using range-mmap-subset strongly-accepts-subset by auto
  qed
qed
```

2.3 Main Theorem

Weirdly, the assumption f ' $\mathcal{F} \subseteq \{..<2::'a\}$ is not used here; it's perhaps unnecessary due to the particular way that *Ramsey* is defined. It's only needed for $(2::'a) < r$

theorem *Nash-Williams-2*:
assumes *thin-set* $\mathcal{F}$ **shows** *Ramsey* $\mathcal{F}$ *2*
unfolding *Ramsey-def*
proof *clarify*
fix f :: *nat set* ⇒ *nat* **and** M :: *nat set*
assume *infinite M*
let $?\mathcal{F} = \lambda i.\ f -` \{i\} \cap \mathcal{F}$
have *finF*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *finite X*
using *assms thin-set-def* **by** *auto*
have *thin*: *thin-set* (*?F i*) **for** *i*
using *assms thin-set-def* **by** *auto*
obtain *N* **where** $N \subseteq M$ *infinite N* **and** *N*: *decides-subsets* (*?F 0*) *N*
using ‹*infinite M*› *ex-infinite-decides-subsets thin* **by** *blast*
then consider *rejects* (*?F 0*) {} *N* | *strongly-accepts* (*?F 0*) {} *N*
unfolding *decides-def decides-subsets-def* **by** *blast*
then show $\exists N\ i.\ N \subseteq M \wedge$ *infinite N* $\wedge\ i<2 \wedge (\forall j<2.\ j \neq i \longrightarrow f -` \{j\} \cap \mathcal{F}$ $\cap$ *Pow N* = {})
proof *cases*
case *1*
then have *?F 0* ∩ *Pow N* = {}
unfolding *rejects-def disjoint-iff*
by (*metis IntD2 PowD comparables-iff finF init-segment-empty sup-bot.left-neutral*)
then show *?thesis*
using ‹*N* ⊆ *M*› ‹*infinite N*› *less-2-cases-iff* **by** *auto*
next
case *2*
then have §: $\bigwedge P.\ [\![P \subseteq N;\ \bigwedge S.\ [\![S \subseteq P;$ *finite S*$]\!] \Longrightarrow S \notin$ (*?F 0*)$]\!] \Longrightarrow$ *finite P*
by (*auto simp*: *Fpow-def disjoint-iff*)
obtain *P* **where** $P \subseteq N$ *infinite P* **and** *P*:
$\bigwedge S\ n.\ [\![S \subseteq P;$ *finite S*; *strongly-accepts* (*?F 0*) *S P*; $n \in P$; *less-sets S* {*n*}$]\!]$
$\Longrightarrow$ *strongly-accepts* (*?F 0*) (*insert n S*) *P*
using *strongly-accepts-1-19-plus* [*OF thin* ‹*infinite N*› *N*] **by** *blast*
have *?F 1* ∩ *Pow P* = {}
proof (*clarsimp simp*: *disjoint-iff*)
fix *T*
assume *T*: *f T* = *Suc 0 T* ∈ $\mathcal{F}$ **and** $T \subseteq P$
then have *finite T*
using *finF* **by** *blast*
moreover have *strongly-accepts* (*?F 0*) *S P* **if** *finite S S* ⊆ *P* **for** *S*
using *that*
proof (*induction card S arbitrary*: *S*)
case (*Suc n*)
then have *Seq*: *S* = *insert* (*Sup S*) (*S* − {*Sup S*})
using *Sup-nat-def Max-eq-iff* **by** *fastforce*

```
      then have card (S − {Sup S}) = n
        using Suc card-Diff-singleton by fastforce
      then have sacc: strongly-accepts (?F 0) (S − {Sup S}) P
        using Suc by blast
      have S ≠ {}
        using Suc.hyps(2) by auto
      have less-sets (S − {Sup S}) {Sup S}
     using Suc.prems(1) finite-Sup-less-iff nat-neq-iff by (auto simp: less-sets-def)
      then have strongly-accepts (?F 0) (insert (Sup S) (S − {Sup S})) P
        by (metis P Seq Suc.prems finite-Diff insert-subset sacc)
      then show ?case
        using Seq by auto
    qed (use 2 ‹P ⊆ N› in auto)
    ultimately have ∃x∈comparables T P. f x = 0 ∧ x ∈ F
      unfolding strongly-accepts-def rejects-def disjoint-iff
      by (metis ‹T ⊆ P› ‹infinite P› IntE order-refl vimage-singleton-eq)
    then show False
      using T assms thin-set-def comparables-def by force
  qed
  then show ?thesis
    by (metis One-nat-def ‹N ⊆ M› ‹P ⊆ N› ‹infinite P› less-2-cases-iff subset-trans)
 qed
qed

theorem Nash-Williams:
 assumes F: thin-set F r > 0 shows Ramsey F r
 using ‹r > 0›
proof (induction r)
 case (Suc r)
 show ?case
 proof (cases r<2)
   case True
   then consider Suc r = 1 | Suc r = 2
     by linarith
   then show ?thesis
   proof cases
     case 1 with F show ?thesis by (auto simp: Ramsey-def)
   next
     case 2 with F show ?thesis by (metis Nash-Williams-2)
   qed
 next
   case False
   with Suc.IH have Ram: Ramsey F r
     by auto
   have r ≥ 2
     by (simp add: False leI)
   show ?thesis
     unfolding Ramsey-def
```

proof *clarify*
fix f **and** M :: *nat set*
assume *fim*: $f \in \mathcal{F} \to \{..<Suc\ r\}$
and *infinite M*
define g **where** $g \equiv \lambda x.\ if\ f\ x = r\ then\ r-1\ else\ f\ x$
have *gim*: $g \in \mathcal{F} \to \{..<r\}$
using *fim False* **by** (*force simp*: *g-def*)
then obtain $N\ i$ **where** $N \subseteq M$ *infinite N* $i<r$ **and** i: $\bigwedge j.\ [\![j<r;\ i\neq j]\!] \Longrightarrow g -` \{j\} \cap \mathcal{F} \cap Pow\ N = \{\}$
using *Ram* ‹*infinite M*› **by** (*metis Ramsey-def*)
show $\exists N\ i.\ N \subseteq M \wedge infinite\ N \wedge i < Suc\ r \wedge (\forall j<Suc\ r.\ j \neq i \longrightarrow f -` \{j\} \cap \mathcal{F} \cap Pow\ N = \{\})$
proof (*cases* $i<r-1$)
case *True*
have $f -` \{j\} \cap \mathcal{F} \cap Pow\ N = \{\}$ **if** $j < Suc\ r\ i \neq j$ **for** j
using ‹$N \subseteq M$› ‹*infinite N*› ‹$i<r$› i *that*
apply (*clarsimp simp add*: *disjoint-iff g-def less-Suc-eq*)
by (*metis True diff-less less-nat-zero-code nat-neq-iff zero-less-one*)
then show *?thesis*
apply (*rule-tac* $x=N$ **in** *exI*)
apply (*rule-tac* $x=i$ **in** *exI*)
by (*simp add*: ‹$N \subseteq M$› ‹$i < r$› ‹*infinite N*› *less-SucI*)
next
case *False*
then have $i = r-1$
using ‹$i<r$› **by** *linarith*
then have *null*: $f -` \{j\} \cap \mathcal{F} \cap Pow\ N = \{\}$ **if** $j<r-1$ **for** j
using *that i* [*of j*] ‹$i < r$› **by** (*auto simp*: *g-def disjoint-iff split*: *if-split-asm*)
define h **where** $h \equiv \lambda x.\ if\ f\ x = r\ then\ 0\ else\ f\ x$
have *him*: $h \in \mathcal{F} \to \{..<r\}$
using *fim i False* ‹$i<r$› **by** (*force simp*: *h-def*)
then obtain $P\ j$ **where** $P \subseteq N$ *infinite P* $j<r$ **and** j: $\forall k<r.\ j\neq k \longrightarrow h -` \{k\} \cap \mathcal{F} \cap Pow\ P = \{\}$
by (*metis Ramsey-def Ram* ‹*infinite N*›)
show *?thesis*
proof (*cases* $j=0$)
case *True*
then show *?thesis*
apply (*rule-tac* $x=P$ **in** *exI*)
apply (*rule-tac* $x=r$ **in** *exI*)
using *null* [*of 0*] ‹$r \geq 2$› ‹$P \subseteq N$› ‹$N \subseteq M$› j ‹*infinite P*›
by (*force simp*: *h-def disjoint-iff less-Suc-eq intro*: *subset-trans*)
next
case *False*
then show *?thesis*
apply (*rule-tac* $x=P$ **in** *exI*)
apply (*rule-tac* $x=j$ **in** *exI*)
using j ‹$P \subseteq N$› ‹$N \subseteq M$› ‹$j < r$› ‹*infinite P*› *False*
by (*auto simp*: *h-def less-Suc-eq disjoint-iff intro*: *less-trans*)

qed
qed
qed
qed
qed *auto*

end

3 Acknowledgements

The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council. Todorčević provided help with the proofs by email.

References

[1] C. S. J. A. Nash-Williams. On well-quasi-ordering transfinite sequences. *Mathematical Proceedings of the Cambridge Philosophical Society*, 61(1):33–39, 1965.

[2] S. Todorčević. *Introduction to Ramsey Spaces*. Princeton University Press, 2010.