The Nash-Williams Theorem

Lawrence C. Paulson

March 17, 2025

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* (**infixr** ‹≪› *50*)
where $A \ll B \equiv \forall x{\in}A.\ \forall y{\in}B.\ x < y$

lemma *less-sets-empty*[*iff*]: $S \ll \{\}\ \{\} \ll T$
by (*auto simp*: *less-sets-def*)

lemma *less-setsD*: $[\![A \ll B;\ a \in A;\ b \in B]\!] \Longrightarrow a < b$
by (*auto simp*: *less-sets-def*)

lemma *less-sets-irrefl* [*simp*]: $A \ll A \longleftrightarrow A = \{\}$
by (*auto simp*: *less-sets-def*)

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

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

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

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

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

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

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

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

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

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* $\Rightarrow$ *nat set* $\Rightarrow$ *bool*
where *init-segment* $S\ T \equiv \exists S'.\ T = S \cup S' \wedge S \ll S'$

lemma *init-segment-subset*: *init-segment* $S\ T \Longrightarrow S \subseteq T$
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* $T\ S]\!] \Longrightarrow S{=}T$
by (*auto simp*: *init-segment-def*)

lemma *init-segment-trans*: $[\![$*init-segment* $S\ T$; *init-segment* $T\ U]\!] \Longrightarrow$ *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* $S\ \{\} \longleftrightarrow S{=}\{\}$
by (*auto simp*: *init-segment-def less-sets-def*)

lemma *init-segment-Un*: $S \ll S' \Longrightarrow$ *init-segment* $S\ (S \cup S')$
by (*auto simp*: *init-segment-def less-sets-def*)

lemma *init-segment-iff0*:
shows *init-segment* $S\ T \longleftrightarrow S \subseteq T \wedge S \ll (T{-}S)$
by (*smt* (*verit*) *DiffD1 DiffD2 Diff-partition UnE init-segment-def init-segment-subset less-sets-def*)

lemma *init-segment-iff*:
shows *init-segment* $S\ T \longleftrightarrow S{=}T \vee (\exists m \in T.\ S = \{n \in T.\ n < m\})$ (**is** *?lhs=?rhs*)
proof
assume *?lhs*
then obtain S' **where** S': $T = S \cup S'$ $S \ll S'$
by (*meson init-segment-def*)
then have $S \subseteq T$
by *auto*
then have *eq*: $S' = T{-}S$
using S' **by** (*auto simp*: *less-sets-def*)
show *?rhs*
proof (*cases* $T \subseteq S$)

case *True*
with ‹$S \subseteq T$› **show** *?rhs* **by** *blast*
next
case *False*
then have *Inf* $S' \in T$
by (*metis Diff-eq-empty-iff Diff-iff Inf-nat-def1 eq*)
moreover have $\bigwedge x.\ x \in S \Longrightarrow x < Inf\ S'$
using S' *False* **by** (*metis Diff-eq-empty-iff Inf-nat-def1 eq less-sets-def*)
moreover have $\{n \in T.\ n < Inf\ S'\} \subseteq S$
using *Inf-nat-def eq not-less-Least* **by** *fastforce*
ultimately show *?rhs*
using ‹$S \subseteq T$› **by** *blast*
qed
next
assume *?rhs*
then show *?lhs*
proof (*elim disjE bexE*)
fix m
assume m: $m \in T\ S = \{n \in T.\ n < m\}$
then have $T = S \cup \{n \in T.\ m \leq n\}$
by *auto*
moreover have $S \ll \{n \in T.\ m \leq n\}$
using m **by** (*auto simp*: *less-sets-def*)
ultimately show *init-segment S T*
using *init-segment-Un* **by** *force*
qed (*use init-segment-refl* **in** *blast*)
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*: $S \ll \{n\}$ **and** *TS*: $\bigwedge x.\ x \in T - S \Longrightarrow n \leq x$
shows *init-segment* (*insert n S*) $T \longleftrightarrow$ *init-segment S T* $\wedge$ $n \in T$
using *DiffD1 Sn TS init-segment-iff0 less-sets-def order-less-le* **by** *fastforce*

lemma *init-segment-insert*:
assumes *init-segment S T* **and** T: $T \ll \{n\}$
shows *init-segment S* (*insert n T*)
by (*metis assms init-segment-Un init-segment-trans insert-is-Un sup-commute*)

2.2 Definitions and basic properties

definition *Ramsey* :: [*nat set set*, *nat*] $\Rightarrow$ *bool*
where *Ramsey* $\mathcal{F}\ r \equiv \forall f \in \mathcal{F} \to \{..<r\}.$
$\forall M.$ *infinite* $M \longrightarrow$
$(\exists N\ i.\ N \subseteq M \wedge$ *infinite* $N \wedge i<r \wedge$
$(\forall j<r.\ j \neq i \longrightarrow f\ -`\ \{j\} \cap \mathcal{F} \cap Pow\ N = \{\}))$

Alternative, simpler definition suggested by a referee.

lemma *Ramsey-eq*:
Ramsey $\mathcal{F}$ $r \longleftrightarrow (\forall f \in \mathcal{F} \to \{..<r\}.$
$\forall M.$ *infinite* $M \longrightarrow$
$(\exists N\ i.\ N \subseteq M \wedge$ *infinite* $N \wedge i<r \wedge \mathcal{F} \cap$ *Pow* $N \subseteq f\ -`$
$\{i\}))$
unfolding *Ramsey-def*
by (*intro ball-cong all-cong ex-cong1 conj-cong refl*) *blast*

definition *thin-set* :: *nat set set* $\Rightarrow$ *bool*
where *thin-set* $\mathcal{F} \equiv \mathcal{F} \subseteq$ *Collect finite* $\wedge$ $(\forall S{\in}\mathcal{F}.\ \forall T{\in}\mathcal{F}.$ *init-segment* $S\ T \longrightarrow S{=}T)$

definition *comparables* :: *nat set* $\Rightarrow$ *nat set* $\Rightarrow$ *nat set set*
where *comparables* $S\ M \equiv \{T.$ *finite* $T \wedge$ (*init-segment* $T\ S \vee$ *init-segment* $S\ T \wedge T{-}S \subseteq M)\}$

lemma *comparables-iff*: $T \in$ *comparables* $S\ M \longleftrightarrow$ *finite* $T \wedge$ (*init-segment* $T\ S \vee$ *init-segment* $S\ T \wedge T \subseteq S \cup M)$
by (*auto simp*: *comparables-def init-segment-def*)

lemma *comparables-subset*: $\bigcup$(*comparables* $S\ M) \subseteq S \cup M$
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*: $N \subseteq M \Longrightarrow$ *comparables* $S\ N \subseteq$ *comparables* $S\ M$
by (*auto simp*: *comparables-def*)

definition *rejects* $\mathcal{F}\ S\ M \equiv$ *comparables* $S\ M \cap \mathcal{F} = \{\}$

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*; $N \subseteq M$⟧ ⟹ *rejects* $\mathcal{F}$ *S N*
by (*fastforce simp add*: *rejects-def comparables-def*)

lemma *strongly-accepts-subset*: ⟦*strongly-accepts* $\mathcal{F}$ *S M*; $N \subseteq M$⟧ ⟹ *strongly-accepts* $\mathcal{F}$ *S N*
by (*auto simp*: *strongly-accepts-def*)

lemma *decides-subset*: ⟦*decides* $\mathcal{F}$ *S M*; $N \subseteq M$⟧ ⟹ *decides* $\mathcal{F}$ *S N*
by (*meson decides-def rejects-subset strongly-accepts-subset*)

lemma *decides-subsets-subset*: ⟦*decides-subsets* $\mathcal{F}$ *M*; $N \subseteq M$⟧ ⟹ *decides-subsets* $\mathcal{F}$ *N*
by (*meson decides-subset decides-subsets-def subset-trans*)

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

lemma *strongly-accepts-empty* [*simp*]: *strongly-accepts* $\mathcal{F}$ {} *M* ⟷ (∀ *N*⊆*M*. *Fpow N* ∩ $\mathcal{F}$ = {} ⟶ *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* ⋀*T*. $T \subseteq S$ ⟹ *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*′)
define Φ **where** Φ ≡ λ*n N*′. *N*′ ⊆ *F n* ∧ *infinite N*′ ∧ *decides* $\mathcal{F}$ (*f* (*Suc n*)) *N*′
have *P-Suc*: *F* (*Suc n*) = (@*N*′. Φ *n N*′) **for** *n*
by (*auto simp*: *F-def* Φ*-def*)
have ∗: *infinite* (*F n*) ∧ *decides* $\mathcal{F}$ (*f n*) (*F n*) ∧ *F n* ⊆ *M* **for** *n*
proof (*induction n*)
case (*Suc n*) **then show** *?case*
by (*metis P-Suc* Φ*-def ex-infinite-decides-1 someI-ex subset-trans*)

```
  qed (auto simp: F-def M0)
  then have telescope: F (Suc n) ⊆ F n for n
    by (metis P-Suc Φ-def ex-infinite-decides-1 someI-ex)
  let ?N = ⋂ 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: (⋂ n < Suc m. F n) = F m for m
      by (induction m) (use telescope in ‹auto simp: lessThan-Suc›)
    then show infinite ?N
      by (metis (full-types) ∗ Pow-top Suc-le-D Suc-le-eq f imageE lessThan-iff)
  next
    fix T
    assume T ⊆ S then show decides F T ?N
      by (metis (no-types) ∗ INT-lower Pow-iff decides-subset f imageE)
  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 F infinite M
  obtains N where N ⊆ M infinite N decides-subsets F N
proof −
  obtain M0 where M0: infinite M0 M0 ⊆ M decides F {} M0
    by (meson ‹infinite M› ex-infinite-decides-1)
  define decides-all where decides-all ≡ λS N. ∀ T ⊆ S. decides F T N
  define Φ where Φ ≡ λNL 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 decides-all-def
      by (metis List.finite-set ex-infinite-decides-finite ‹infinite (hd NL)›)
    then have inf: infinite (N ∩ {Inf (hd NL)<..})
      by (metis finite-nat-Int-greaterThan-iff)
    then have Inf (N ∩ {Inf (hd NL)<..}) > Inf (hd NL)
      by (metis finite.emptyI Inf-nat-def1 Int-iff greaterThan-iff)
    with N show ?thesis
      unfolding Φ-def
      by (meson Int-lower1 decides-all-def decides-subset inf 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] (λn 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)
  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 Suc-le-mono ‹finite K› hd-in-set imageI map-Inf-subset
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 (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 hd-Suc-eq-Eps less-imp-le-nat subsetD
that)
 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 ‹strict-mono mmap› finite-imageD strict-mono-on-imp-inj-on 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 ∃ m. S ⊆ List.set (map Inf (F m))
    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 S ≪ {x} for x
      proof clarsimp
        fix T
        assume x = Inf T and T ∈ list.set (F n)
        with that have ls: 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
          moreover have Eps (Φ (F j)) ⊆ T
            by (metis F hd-Suc-eq-Eps sorted-wrt-subset that)
          ultimately have k < j
                unfolding mmap-def by (metis Inf-hd-in-Eps ‹x ∉ T› in-mono
not-less-eq)
          then show x ∈ Inf ‘ list.set (F j)
            using Suc-leI hd-F-in-F k mmap-def by blast
        qed
        then show False
          by (metis False ‹T ∈ set (F n)› lessI minS not0-implies-Suc)
      qed
      have Inf-hd-F: Inf (hd (F m)) ∈ Eps (Φ (F n)) if 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: 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
        then have rejects F S (range mmap)
       apply (simp add: rejects-def disjoint-iff mmap-def comparables-def image-iff
subset-iff)
          by (metis less-S Inf-hd-F hd-Suc-eq-Eps)
        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 ‹infinite N›)
        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 show ?thesis
        by (auto simp: decides-def strongly-accepts-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
    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 ¬ init-segment T S
    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 *TS*: *finite* (*T*−*S*) *T* − *S* ≠ {}
using *T*(*1*) *init-segment-subset* **by** (*force simp*: *comparables-iff*)+
then have *?n* ∈ *N*
by (*meson Diff-subset-conv Min-in TSN subsetD*)
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* ‹*?n* ∈ *N*› ‹*finite S*›
by (*auto simp*: *rejects-def comparables-iff disjoint-iff*)
moreover have *S* ≪ {*?n*}
using *Sup-nat-less-sets-singleton N* ‹*?n* ∈ *N*› ‹*finite S*› **by** *blast*
ultimately show *?thesis*
using *2* **by** (*metis DiffD1 eq-Min-iff TS init-segment-insert-iff*)
qed
qed

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

proposition *strongly-accepts-1-19-plus*:
assumes *thin-set* $\mathcal{F}$ *infinite M*
and *dsM*: *decides-subsets* $\mathcal{F}$ *M*
obtains *N* **where** *N* ⊆ *M infinite N*
⋀*S n*. ⟦*S* ⊆ *N*; *finite S*; *strongly-accepts* $\mathcal{F}$ *S N*; *n* ∈ *N*; *S* ≪ {*n*}⟧
⟹ *strongly-accepts* $\mathcal{F}$ (*insert n S*) *N*
proof −
define *insert-closed* **where**
insert-closed ≡ λ*NL N*. ∀ *T* ⊆ *Inf* ‘ *set NL*. ∀ *n* ∈ *N*.
strongly-accepts $\mathcal{F}$ *T* ((*Inf* ‘ *set NL*) ∪ *hd NL*) ⟶
T ≪ {*n*} ⟶ *strongly-accepts* $\mathcal{F}$ (*insert n T*) ((*Inf* ‘ *set NL*) ∪ *hd NL*)
define Φ **where** Φ ≡ λ*NL 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* ≡ λ*S*. {*n* ∈ *?M*. ¬ *strongly-accepts* $\mathcal{F}$ (*insert n S*) *?M*}
have ∃ *k*. *P S* ⊆ {..*k*}
if *S* ⊆ *Inf* ‘ *set NL strongly-accepts* $\mathcal{F}$ *S ?M* **for** *S*
proof −
have *decides-subsets* $\mathcal{F}$ *?M*
using *NL*(*2*) *decides-subsets-subset dsM* **by** *blast*

```
    with that NL assms finite-surj have finite (P S)
      unfolding P-def by (blast intro!: strongly-accepts-1-19)
    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)
    show infinite (hd NL ∩ {m<..})
      by (simp add: finite-nat-Int-greaterThan-iff that(1))
    moreover have Inf (hd NL) ≤ m
      by (simp add: m-def)
    ultimately show Inf (hd NL) < Inf (hd NL ∩ {m<..})
      using Inf-nat-def1 [of (hd NL ∩ {m<..})] by force
    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 ‘ set (F n) ⊆ M for n
proof (induction n)
  case (Suc n)
  have hd (F n) ⊆ M
    by (meson Pow-iff Suc.IH hd-in-set subsetD)
  then obtain Φ: Ball (list.set (F n)) ((⊆) (Eps (Φ (F n)))) infinite (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*)
then have *M*: *Eps* (Φ (*F n*)) ⊆ *M*
by (*meson Pow-iff Suc.IH hd-in-set subset-iff*)
with Φ **have** *Inf* (*Eps* (Φ (*F n*))) ∈ *M*
by (*metis Inf-nat-def1 finite.simps in-mono*)
with Φ *M* **show** *?case*
using *Suc* **by** *simp*
qed (*auto simp*: *InfM* ‹*infinite M*›)
have Φ*F*: Φ (*F n*) (*Eps* (Φ (*F n*))) **for** *n*
by (*metis Ball-Collect F Pow-iff Un-subset-iff* Φ*-Eps hd-in-set subsetD*)
then have *insert-closed*: *insert-closed* (*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 Φ*F* Φ*-def* **by** *blast*
define *mmap* **where** *mmap* ≡ λ*n*. *Inf* (*hd* (*F* (*Suc n*)))
have *mmap n* < *mmap* (*Suc n*) **for** *n*
by (*metis F-simps*(*2*) Φ*F* 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* (Φ (*F* (*Suc n*))) ⊆ *Eps* (Φ (*F n*)) **for** *n*
by (*metis F-simps*(*2*) Φ*F* 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* (*F n*)) ∈ *hd* (*F n*) **for** *n*
by (*metis Inf-nat-def1* Φ*F* Φ*-def finite.emptyI 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 hd-Suc-eq-Eps nat-less-le subsetD that*)
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 image-mmap-subset-hd-F* **by** *fastforce*
next
fix *S a*
assume *S*: *S* ⊆ *range mmap finite S* **and** *acc*: *strongly-accepts* ℱ *S* (*range mmap*)
and *a*: *a* ∈ *range mmap* **and** *Sn*: *S* ≪ {*a*}

```
    then obtain n where n: a = mmap n
      by auto
    define N where N ≡ LEAST n. S ⊆ mmap ‘ {..<n}
    have ∃ n. S ⊆ mmap ‘ {..<n}
      by (metis 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
    qed (auto simp: Sn)
    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

lemma *Nash-Williams-1*: *Ramsey* $\mathcal{F}$ *1*
by (*auto simp*: *Ramsey-eq*)

theorem *Nash-Williams-2*:
assumes *thin-set* $\mathcal{F}$ **shows** *Ramsey* $\mathcal{F}$ *2*
unfolding *Ramsey-eq*
proof *clarify*
fix f :: *nat set* ⇒ *nat* **and** M :: *nat set*
assume *infinite M* **and** *f2*: $f \in \mathcal{F} \to \{..<2\}$
let $?\mathcal{F} = \lambda i.\ f -` \{i\} \cap \mathcal{F}$ — needed with *Ramsey-eq*, not with *Ramsey-def*
have $\mathcal{F}$: $?\mathcal{F}\ 0 \cup ?\mathcal{F}\ 1 = \mathcal{F}$
using *f2 less-2-cases* **by** (*auto simp*: *PiE*)
have *finF*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *finite X* **and** *thin*: $\bigwedge i.$ *thin-set* $(?\mathcal{F}\ i)$
using *assms thin-set-def* **by** *auto*
then obtain N **where** $N \subseteq M$ *infinite N* **and** N: *decides-subsets* $(?\mathcal{F}\ 0)\ N$
using ‹*infinite M*› *ex-infinite-decides-subsets* **by** *blast*
then consider *rejects* $(?\mathcal{F}\ 0)\ \{\}\ N \mid$ *strongly-accepts* $(?\mathcal{F}\ 0)\ \{\}\ N$
unfolding *decides-def decides-subsets-def* **by** *blast*
then show $\exists N\ i.\ N \subseteq M \wedge$ *infinite* $N \wedge i<2 \wedge \mathcal{F} \cap$ *Pow* $N \subseteq f -` \{i\}$
proof *cases*
case *1*
then have $(?\mathcal{F}\ 0 \cup ?\mathcal{F}\ 1) \cap$ *Pow* $N \subseteq f -` \{1\}$
using *f2 finF* **by** (*auto simp*: *Fpow-Pow-finite*)
then show *?thesis*
by (*metis* $\mathcal{F}$ *Suc-1* ‹$N \subseteq M$› ‹*infinite N*› *lessI*)
next
case *2*
then have §: $\bigwedge P.\ [\![P \subseteq N;\ \bigwedge S.\ [\![S \subseteq P;$ *finite* $S]\!] \Longrightarrow S \notin ?\mathcal{F}\ 0]\!] \Longrightarrow$ *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* $(?\mathcal{F}\ 0)\ S\ P;\ n \in P;\ S \ll \{n\}]\!]$
$\Longrightarrow$ *strongly-accepts* $(?\mathcal{F}\ 0)$ *(insert n S) P*
using *strongly-accepts-1-19-plus* [*OF thin* ‹*infinite N*› *N*] **by** *blast*
have $\mathcal{F} \cap$ *Pow* $P \subseteq f -` \{0\}$
proof (*clarsimp simp*: *subset-vimage-iff*)
fix T
assume T: $T \in \mathcal{F}$ **and** $T \subseteq P$
then have *finite T*
using *finF* **by** *blast*
moreover have *strongly-accepts* $(?\mathcal{F}\ 0)\ S\ P$ **if** *finite* S $S \subseteq P$ **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 *sacc*: *strongly-accepts* $(?\mathcal{F}\ 0)\ (S - \{$*Sup S*$\})\ P$
by (*metis Suc card-Diff-singleton diff-Suc-1 finite-Diff insertCI insert-subset*)

```
      have S − {Sup S} ≪ {Sup S}
      using Suc by (simp add: Sup-nat-def dual-order.strict-iff-order 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
      using ‹T ⊆ P› ‹infinite P› rejects-def strongly-accepts-def by fastforce
    then show f T = 0
      using T assms thin-set-def comparables-def by force
  qed
  then show ?thesis
    by (meson ‹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
   with less-2-cases Nash-Williams-1 Nash-Williams-2 assms show ?thesis
     by (auto simp: numeral-2-eq-2)
 next
   case False
   with Suc.IH have Ram: Ramsey F r r ≥ 2
     by auto
   show ?thesis
     unfolding Ramsey-eq
   proof clarify
     fix f and M :: nat set
     assume fim: f ∈ F → {..<Suc r}
       and infinite M
     let ?within = λg i N. F ∩ Pow N ⊆ g −‘ {i}
     define g where g ≡ λx. if f x = r then r−1 else f x
     have gim: g ∈ F → {..<r}
       using fim False by (force simp: g-def)
     then obtain N i where N ⊆ M infinite N i<r and i: ?within g i N
       using Ram ‹infinite M› by (metis Ramsey-eq)
     show ∃N j. N ⊆ M ∧ infinite N ∧ j < Suc r ∧ ?within f j N
     proof (cases i<r−1)
       case True
       then have ?within f i N
         using ‹N ⊆ M› ‹infinite N› ‹i<r› i by (fastforce simp add: g-def)
```

```
      then show ?thesis
        by (meson ‹N ⊆ M› ‹i < r› ‹infinite N› less-Suc-eq)
    next
      case False
      then have i = r−1
        using ‹i<r› by linarith
      then have null: ℱ ∩ Pow N ⊆ f −‘ {i,r}
        using i ‹i < r›
        by (auto simp: g-def split: if-split-asm)
      define h where h ≡ λx. if f x = r then 0 else f x
      have him: h ∈ ℱ → {..<r}
        using fim i False ‹i<r› by (force simp: h-def)
      then obtain P j where P ⊆ N infinite P j<r and j: ?within h j P
        using Ram ‹i < r› ‹infinite N› unfolding Ramsey-eq by metis
      have ∃ i < Suc r. ?within f i P
      proof (cases j=0)
        case True
        then have ℱ ∩ Pow P ⊆ f −‘ {r}
          using Ram(2) ‹P ⊆ N› ‹i = r − 1›  i j
          unfolding subset-vimage-iff g-def h-def
           by (metis Int-iff Pow-iff Suc-1 diff-is-0-eq insert-iff not-less-eq-eq sub-
set-trans)
        then show ?thesis
          by blast
      next
        case False
        then show ?thesis
          using j ‹j < r› by (fastforce simp add: h-def less-Suc-eq)
      qed
      then show ?thesis
        by (meson ‹N ⊆ M› ‹P ⊆ N› ‹infinite P› subset-trans)
    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.